Final Technical Report
for DARPA Contract HR0011-04-1-0005

KI-LEARN: Knowledge-Intensive Learning Methods for Knowledge-Rich/Data-Poor Domains

Authors: Thomas G. Dietterich, Professor; Angelo Restificar, Research Associate; Prasad Tadepalli, Associate Professor; Bruce D'Ambrosio, Associate Professor; Jon Herlocker, Assistant Professor; Alan Fern, Assistant Professor; Eric Altendorf, Graduate Student; Sriraam Natarajan, Graduate Student; Jianqiang Shen, Graduate Student; Xinlong Bao, Graduate Student; Anton Dragunov, Graduate Student

Performing Organization: Oregon State University, Corvallis, OR 97331

Date: 31 August 2006

Type of Report: Final

Period Covered: 10/29/2003 - 09/30/2005

ABSTRACT

Knowledge Representation and Reasoning (KRR) has developed a wide range of methods for representing knowledge and reasoning from it to produce expert-level performance. Despite these accomplishments, there is one major problem preventing the wide-spread application of KRR technology: the inability to support learning. This makes KRR systems brittle and difficult to maintain. On the other hand, Machine Learning (ML) has developed a wide range of methods for learning from examples. However, there are two major problems preventing the wide-spread application of machine learning technology: the need for large amounts of training data and the high cost of manually designing the hypothesis space of the learning system. Our goal in this research effort was to develop a new methodology, called KI-LEARN (**K**nowledge **I**ntensive **LEARN**ing), that combines domain knowledge and sparse training data to construct high-performance systems. This report provides an overview of the major results we obtained on specific tasks as outlined in our proposal. More specifically, to address issues in knowledge representation and efficient learning we designed a language called First-Order Conditional Influence (FOCI) Language for expressing attributes relevant to learning. Our language extends probabilistic relational models (PRMs) which are themselves probabilistic representations most similar to first-order representation languages employed in KRR systems. A distinct feature of our language is its support for explicit expression of qualitative constraints such as monotonicity, saturation, and synergies. More importantly, we have demonstrated via mathematical proofs and experimental results how these qualitative constraints can be used and exploited when learning with sparse training data. We specifically show how qualitative constraints can be incorporated into learning algorithms. In addition, this report describes the models we constructed for our testbed domains. We also describe the infrastructure we built for the task-based user interface domain as well as further improvements we made to the software prototype. Finally, we provide a list of research publications that were produced either in part or solely through the KI-LEARN research grant.

Contents

1 Introduction

Knowledge Representation and Reasoning (KRR) has developed a wide range of methods for representing knowledge and reasoning from it to produce expert-level performance [15]. These representations generally cover substantial subsets of first-order predicate logic. In addition, they often provide special vocabularies and reasoning methods for reasoning about time, space, action, qualitative relationships, communication, and so on. Despite these accomplishments, there is one major problem preventing the wide-spread application of KRR technology: **the inability to support learning**. This makes KRR systems brittle and difficult to maintain.

Machine Learning (ML) has developed a wide range of methods for learning from examples. These include methods for classification (predicting a discrete-valued variable) and regression (predicting a real-valued variable), as well as methods for handling sequential and spatial data. Recent work has begun to develop methods for learning in relational models [31, 21]. Despite these accomplishments, there are two major problems preventing the wide-spread application of machine learning technology: **the need for large amounts of training data and the high cost of manually designing the hypothesis space of the learning system**.

Existing machine learning systems require large amounts of training data, because they generally start without any knowledge of the application problem. There is a fundamental relationship in all learning systems between the amount of available training data and the amount of the knowledge that can be learned [7, 8]. Large amounts of training data are needed in order to learn large amounts of knowledge. If only a small amount of data is available, then only small amounts of learning can be supported.

As a result, the machine learning community generally prefers to study problems where there are massive amounts of data collected. This has yielded major progress in mining massive data sets for scientific and business applications. But many important military and civilian problems have the property that there is relatively little data available. For example, an important potential application area is the development of cognitive assistants for human-computer systems. A *tabula rasa* approach to machine learning for such assistive systems would require the human user to provide many many hours of labeled training data to the learning system. This is completely impractical. Another important application area is early-phase epidemiology: understanding the spread of new diseases (including diseases introduced as weapons of terror). During the early phases of a new disease, rapid learning is essential in order to understand and halt the epidemic, but very little data is available.

Because the training data is so sparse, these kinds of problems are largely ignored by the machine learning community. This is because the only available methodology for handling knowledge-rich/ data-poor problems is for the data analyst to study the domain knowledge and design a special-purpose learning system that incorporates this knowledge. To make these systems effective, the data analyst must carefully design the set of features (i.e., the attributes that describe the data) and the space of hypotheses (i.e., the representations that use those attributes) so that it is small enough. This is a time-consuming process, and there are relatively few data analysts who can do it. There are also very few software tools available to support this activity. Existing off-the-shelf machine learning systems do not provide any way to incorporate background knowledge except through defining the input features.

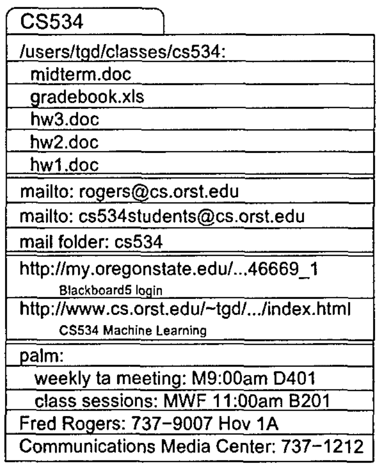
Our new methodology enables a wide range of new cognitive system applications in problems that require combining domain knowledge with training data. These include cognitive assistants (e.g., for computer users, war fighters, and counter-terrorism analysts) and situation awareness/situation assessment systems (e.g., for infectious disease spread, bio-terrorism, and command-and-control). This report provides a detailed discussion on the methodology we have formulated for building applications in knowledge-rich/data-poor domains. It also provides a discussion of the problems we have encountered and a summary of the major results we have obtained thus far.

2 Methods and Procedures

To construct cognitive systems in knowledge-rich/data-poor domains we followed a set of steps that allowed us (1) to design practical systems that could exploit existing domain knowledge using representations that are sufficiently expressive for learning with sparse data and (2) to test and evaluate the effectiveness of such design. We applied these steps on various domains: (i) task-based user interfaces (ii) modeling the spread of West Nile Virus, and (iii) data sets obtained from the UC-Irvine repository. We first give a description of the domains and data sets we have used, then we describe the set of steps we have used to construct systems in knowledge-rich/data-poor domains.

2.1 Description of Test-Bed Domains

The following provides an overview of the domains we have used for problem analysis, algorithm design, and evaluation of the techniques we have formulated in this research effort.



CS534
/users/tgd/classes/cs534:
midterm.doc
gradebook.xls
hw3.doc
hw2.doc
hw1.doc
mailto: rogers@cs.orst.edu
mailto: cs534students@cs.orst.edu
mail folder: cs534
http://my.oregonstate.edu/...46669_1
Blackboard5 login
http://www.cs.orst.edu/~tgd/.../index.html
CS534 Machine Learning
palm:
weekly ta meeting: M9:00am D401
class sessions: MWF 11:00am B201
Fred Rogers: 737-9007 Hov 1A
Communications Media Center: 737-1212

Figure 1: Example menu of documents relevant to the CS534 task

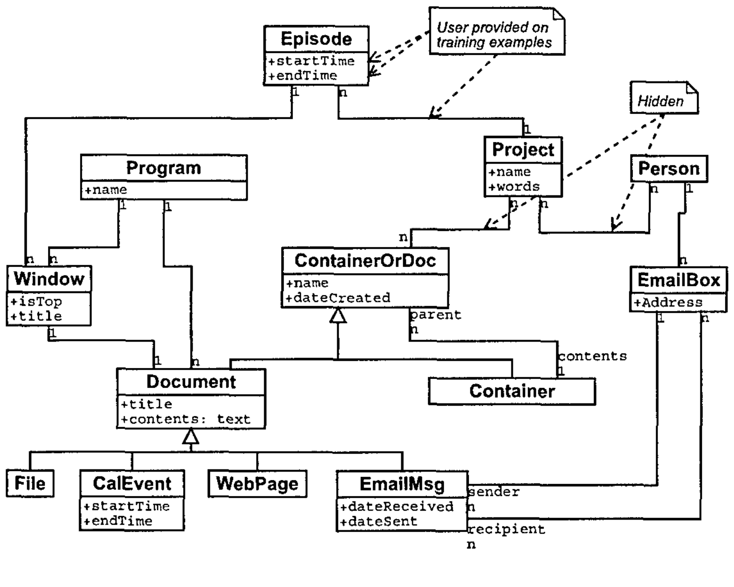


Figure 2: UML Object-Relational Schema for human-computer interface

2.1.1 Task-Based User Interfaces

Imagine a human-computer interface that learns to recognize the tasks that you are performing and finds ways to help you. For example, if you are a professor and you open the file /users/tgd/classes/cs534/midterm.doc, the interface could recognize that you are working on your CS534 class. It could pop up a menu of documents and contacts relevant to that task (see Figure 1) that includes such items as the course gradebook, the email address and phone number of the teaching assistant, course web sites, and scheduled meetings. It could even start up other relevant applications *e.g.,* pre-fetch the class web page into your web browser; pre-load the gradebook spreadsheet into excel. To make this kind of user interface work, we need a learning system that can learn to recognize your current task based on a small amount of training data. For example, the first few times that you work on CS534, you need to tell the computer that CS534 is the current task. But later, it should quickly recognize the current task without being told. This is exactly the kind of knowledge-rich/data-poor machine learning problem for which existing methods do not work well.

2.1.2 West Nile Virus Propagation

A significant threat to public health is the spread of infectious diseases. These may arise naturally or be introduced as bioweapons. The spread of West Nile Virus provides a useful case study. In previous research, one of us (D'Aambrosio, with several collaborators) has developed a community-level model of the disease transmission process of West Nile Virus (WNV) in Maryland [47]. In this domain, most of the background knowledge takes the form of a qualitative model of the life cycles of various organisms including the virus, mosquitos, corvid birds (crows and their relatives), horses, and people. The available data (covering 1999-2002) consists of the following: (a) date and location of human, horse, and bird cases, (b) test results of mosquitos in pools, (c) test results of mosquitos captured in traps, and (d) locations of tire collection and recycling facilities. The tire information is relevant because pools of water form frequently in tires and mosquito larvae live in pools. The development of a complete epidemiological model for a disease takes many years of study. For bio-defense purposes, however, there is a need to learn an initial model very quickly from small amounts of training data.

2.1.3 UC-Irvine Data Sets

We have chosen ten data sets from the UC-Irvine Machine Learning Repository. These data sets were chosen based on their potential to be used as test-beds for learning algorithms that exploit domain knowledge specifically in the form of monotonic relationships between attributes *e.g.,* an increase in body-mass index leads to higher risk of having elevated blood pressure. A substantial amount of time was devoted to examining these data sets so that graphical models, in this case Bayesian Networks, that reflect as closely as possible what a domain expert might construct were elicited. The elicitations were based on common general knowledge and existing documents that either came with the data sets or were cited by their respective accompanying documentations. In addition, these documents were carefully studied to identify all indications of possible monotonic relationships. This information is then used to annotate the previously-elicited network structure. As often happens in real-world modeling, we consider both the network structure and the knowledge about monotonic relationships between the variables in these networks to be only approximately correct.

2.2 Building Domain Knowledge Bases

We have constructed domain knowledge bases using varying levels of representations: both propositional and relational. Representations at the propositional level are derived from simple statements, *i.e.,* non-relational statements, about the domain and its relevant attributes or events. These statements could express causality between events and additionally, could indicate qualitative relationship between attributes (via annotations), such as monotonicity or saturation. Representations at the relational level are in terms of an object-relational schema describing the ontology of the domain (the classes of objects and their attributes and relations). These also include representations of relationships among object attributes including qualitative causality and probability statements, for instance as in the propositional representation, a monotonic relationship between random variables. The domain knowledge at this level also describes which objects, attributes, and relations are observable and which are hidden. In addition, the knowledge base specifies the attributes and relations for which it is desired to learn a predictive model. These are called the "targets" of the learning process.

2.3 Defining Features for Learning

The constructed domain knowledge is extended to include new features more suitable for learning. Some existing attributes may make excellent features, but it is typical in machine learning applications to construct new features by (a) aggregating existing attributes *e.g.,* over temporal and spatial scales to reduce noise and improve statistical power, and (b) transforming attributes *e.g.,* by Fourier or wavelet transforms and principal component analysis to enhance pattern detection.

It is also possible to exploit domain knowledge so that feature design reflects the choice of the set of features that are only considered relevant for learning. For example in the domain of task-based user interfaces, consider the attributes of a document. A document has content (the words in the document), but it also has a length (in bytes), a creation date, the program used to create it, and so on. The domain knowledge indicates that the content is likely to be more useful for predicting the current project than these additional attributes. A purely syntactic approach to feature generation would generate these kinds of irrelevant features and then require large amounts of training data to eliminate them. In the case of task-based user interfaces, domain rules can be exploited to help choose relevant features.

To give a more concrete example consider Figure 2 which shows a schema for user interaction with a computer system. Each box represents a class of object with the indicated attributes. Each link represents a relationship between two classes. For example, every Window on the computer screen belongs to some Program and corresponds to some Document. In this schema, files, web pages, email messages, and calendar events are all considered Documents. The schema describes the user's interaction with the computer as a sequence of Episodes. Each Episode corresponds to some Project, and the goal of the learning system is to learn to identify the Project of each Episode early in that episode, so that the interface can pop up the relevant documents menu. Each Document may belong to one or more Projects. Documents may also be in Containers, which is a generalization of file directories, web sites, mail folders, etc. EmailMsgs have senders and receivers, which are EmailBoxes, and these in turn are owned by Persons. A Person may belong to one or more Projects. In the task-based user interface domain the following domain knowledge rules could help choose relevant features.

- DRule1: Most Documents touched during an Episode belong to the episode's Project.
- DRule2: Documents in the same Container tend to share a common Project.
- DRule3: Documents on the same Project tend to share common words in their contents, title, name, email subject, etc.

DRule 1 tells us that we can predict the project of an episode if we know the project of a document that was in the top-most window during the episode. DRule 2 tells us that we can predict the project of a document by looking at the other documents in the same container. DRule 3 tells us that we can predict the project of a document by looking at other documents with similar contents, title, name, etc. We can combine these rules to obtain "paths" through the schema that would be useful for predicting the project of the current episode. Each path can be written as an expression in a path language, and it denotes a bag of objects that are reachable along the path. For example, if we combine rules 1 and 2, we obtain the path

Window[isTop].Document.Container.ContainedDocuments.Project

which describes the bag of all projects such that they belong to a document that is in the same container as the document of the top window. The expression [isTop] is a selector that selects only those windows for which isTop is true.

To convert a bag of projects into a feature for machine learning purposes, we need to summarize it by a fixed vector of features. General machine learning knowledge can be applied here:

- ML Rule 1: A bag over an open domain can be represented by its most common element (An open domain has no bound on the number of possible domain elements.)
- ML Rule 2: A bag over an open domain can be represented by its k most common elements
- ML Rule 3: A bag over a closed domain can be represented by a vector with one entry per element such that entry i specifies the number of times that element i appears in the bag.

In this case, ML Rule 1 would define the feature:

Window[isTop].Document.Container.ContainedDocuments.Project[mostFreq].

In addition to generating candidate features, machine learning knowledge can be applied to select a good subset of them by estimating the informativeness and uniqueness of each feature. A feature is informative if it provides strongly predictive information *e.g.,* about the current Project. A feature is unique if it provides information not provided by other features. Feature informativeness and uniqueness can be crudely estimated by applying such rules (in the case of relational representation) as "A feature defined by a long path will typically be less informative than a feature defined by a short path".

2.4 Extracting Constraints for Learning

We analyzed domain knowledge to identify facts and relationships that can constrain the learning process. The domain knowledge usually provides much more information than can be captured simply by selecting features. For example, DRule 1 tells us not only that the project of the current document is a good feature for predicting the project of the current episode, but also that they are positively correlated. If the project of the current document is Project12, this should increase the probability that the project of the current episode is also Project12. As another example, consider the West Nile Virus problem. The probability of being bitten by an infectious mosquito increases monotonically with the number of infected mosquitos and decreases monotonically with the number of potential hosts (people, horses, birds) for the mosquito byte. We formalized machine learning knowledge for extracting constraints (mutual exclusion, monotonicity, saturation, synergy, anti-synergy) from the domain knowledge. Futhermore, we have extended existing learning algorithms for Bayesian Networks and PRMs to incorporate these constraints.

2.5 Combining Hand-Crafted and Learned Knowledge

Learning establishes quantitative relationships between the chosen features and the targets subject to the constraints. The learned relationships are combined with the domain knowledge to predict the targets for new cases. We validated the knowledge resulting from learning by checking it for consistency with the hand-crafted domain knowledge. Contradictions are detected and resolved either by deleting the hand-crafted rule (if the learned rule has high empirical support) or by deleting the learned rule. Conflicts are typically resolved by first using the hand-crafted rule to suggest additional features for learning and subsequently, repeating the learning step. To combine hand-crafted and learned knowledge, we have extended recent work by ourselves and others on PRMs and Bayesian networks to reduce the gap between representations.

2.6 Software Development for Testing and Evaluation

We have developed software to support our investigative efforts. Computer programs, written principally in Java, C#, and Python, have been used to test and validate our hypotheses. Software development cycles are consistent with the dates indicated for our deliverables.

3 Results and Discussions

In this section, we provide a detailed overview of the results we have obtained as well as the issues and problems we have addressed and encountered during the course of the research effort.

3.1 Knowledge Representation for Efficient Learning

Current knowledge representation and reasoning (KRR) systems cannot adequately support learning in data-sparse domains. Figure 3 shows the standard architecture for a KRR system. The domain knowledge is represented in declarative form (typically as a combination of an ontology, a set of facts, and a collection of rules). This knowledge is processed by an inference engine to solve problems. Notice that there is no way to incorporate training data into such a system.

Figure 4 shows the standard architecture for machine learning (ML) systems. Raw data is preprocessed by a feature extraction subsystem to produce training examples expressed as vectors of feature values. The feature extraction system is typically an ad hoc, hand-crafted program, although some data mining systems provide a higher-level programming language for defining feature extraction procedures. The training examples are then processed by a general purpose machine learning algorithm which outputs the learned knowledge in some form (typically a neural network, decision tree, set of decision rules, or Bayesian network). This output structure is then interpreted by an inference engine to solve problems. Note, however, that this inference engine tends to be much simpler than the inference engines of KRR systems, since it typically does not need to perform any search or complex inference. Also note that there is no way to incorporate knowledge into the architecture.

A natural question that arises is the following: Why not change the representations of ML systems so that they have the same form as the representations employed in KRR systems? This has been the line of research pursued in the Inductive Logic Programming (ILP) community [31]. However, there are two problems with this approach. First, the KRR representations are too expressive to support efficient learning. There is a direct relationship between the expressive power of a representation and the amount of training data needed to learn in that representation. Simple boolean conjunctions over n variables can be accurately learned from approximately $10n$ examples. But arbitrary boolean for-

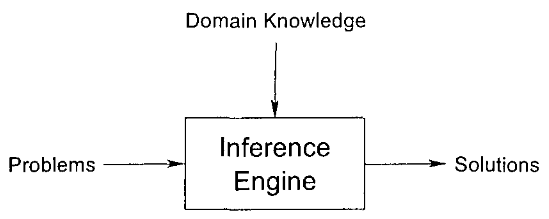


Figure 3: Standard architecture for KRR systems

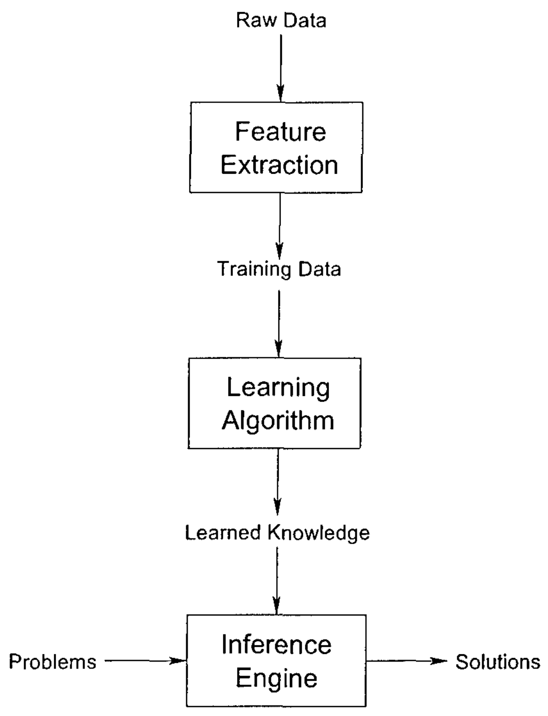


Figure 4: Standard architecture for ML systems

mulas over *n* variables appear to require exponentially-many examples. KRR representations typically include rich subsets of first-order logic, and these would require astronomical numbers of examples for learning. ILP researchers have had to find various ways of constraining their representation languages and search algorithms to make learning feasible, but these constraints are often artificial and unconnected to underlying domain knowledge [29]. Second, KRR inference engines presume that the knowledge given to them is correct. ML inference engines, in contrast, must treat the (learned) knowledge given to them as only approximately correct. This has led ML researchers to adopt probabilistic inference engines and ensemble methods [16], both of which can recover from errors in the learned knowledge.

We have addressed the issue of finding a suitable representation to support efficient learning in two ways. First, we extracted and exploited features and constraints from the hand-crafted knowledge and incorporated them into ML algorithms. The advantage of this approach is that it works with existing ML algorithms and inference engines, whose representations are well-suited to learning. In the propositional case, we have extended fitting algorithms for Bayesian networks so that they can accept a wide range of constraints [1, 40]. We will postpone the discussion of how these constraints are incorporated into machine learning algorithms to Section 3.4.

Second, we designed a new first-order language (and an associated learning algorithm) that we believe will enable the modeler to directly express the domain knowledge in such a way that it can be employed directly in learning, rather than needing to be translated into propositional form and fed to existing ML algorithms. The language is called the *First-Order Conditional Influence* language (FOCI). FOCI is built around standard entity-relationship (ER) models, which allows the user to express the objects, attributes, and relations of the domain. In addition FOCI also supports the expression of probabilistic influences (a form of constraint) between attributes in the ER-model [34, 35]. Our language extends probabilistic relational models (PRMs [22]) which are themselves probabilistic representations most similar to the first-order representation languages employed in KRR systems. FOCI supports qualitative constraints such as monotonicity, saturation and synergies. The primary goal in the language design was not to come up with the most expressive language but to design the most useful and tractable knowledge representation tool suitable for efficient learning.

3.1.1 First-Order Conditional Influence (FOCI) Language

Domain knowledge can be expressed in terms of FOCI statements. Each FOCI statement has an IF-condition which is a conjunction of literals, which when true, defines a set of random variables called "influents" (these are the parents of a random variable in the Bayesian network jargon) that infuence a target variable. The IF-condition generalizes the idea of path expressions in PRMs and makes explicit the difference between logical and random variables that is implicit in Bayesian Logic Programs (BLPs) [28]. The domain expert is assumed to know the structure of the ER diagram while (s)he writes down the FOCI statements. Each FOCI statement has the form:

$$\textit{If } \langle \textit{ condition } \rangle \textit{ then } \langle \textit{ qualitative influence } \rangle$$

where *condition* is a conjunction of literals, each literal being a predicate symbol applied to the appropriate number of variables. Usually, the conditions are used to identify the objects that participate

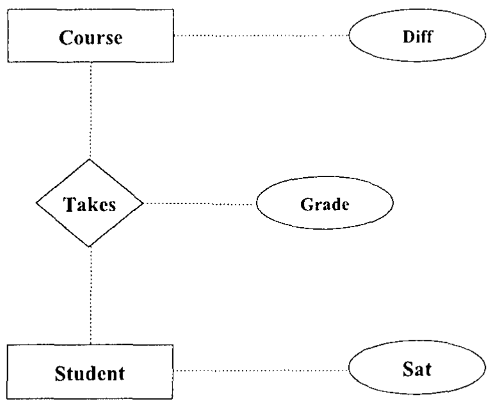


Figure 5: An Example of ER diagram

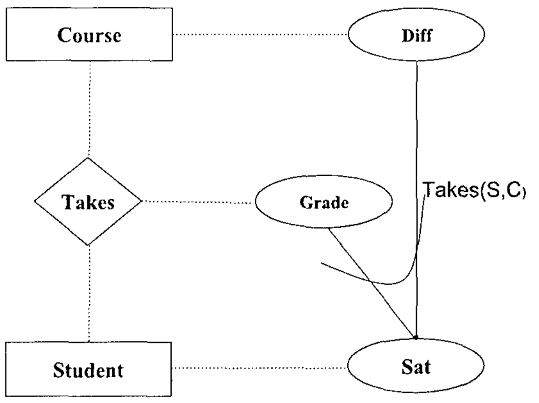


Figure 6: ER Diagram annotated with hyper-arcs for the student satisfaction example

in the qualitative influence. A ⟨*qualitative influence*⟩ is of the form $X_1, \ldots, X_k \overset{Qinf}{\succ} Y$, where the X_i and Y are of the form $V.a$, where V is a variable in *condition* and a is an object attribute. This statement simply expresses a directional dependence of the *resultant* Y on the *influents* X_i. Associated with each FOCI statement is a *conditional probability function* that specifies a probability distribution of the resultant conditioned on the influents, e.g. $P(Y|X_1, \ldots, X_k)$ for the above statement. Consider for example the FOCI statement,

```
If {Person(p)} then p.diettype Qinf p.fitness
```

which indicates that a person's type of diet influences their fitness level. There is an entity `Person` and `diettype` and `fitness` are two attributes of `Person`. As can be seen, the influent and the resultants are of the form `Object.attribute`. Also, the condition is used to determine the type of the variable `p`, which in this case is a `Person`. The conditional probability distribution $P(p.\text{fitness} \mid p.\text{diettype})$ associated with this statement (partially) captures the quantitative relationships between these attributes.

In general, there are several attribute instances that satisfy the IF-condition, and each such tuple has some influence on the target variable. Since there can be a variable number of such instance tuples, we need a way to combine their influences in an efficient manner. Our language allows both aggregators used in the probabilistic relation model literature and combining rules (such as noisy-OR) to compactly specify such joint influences. In addition, we can combine the effects of different FOCI statements on a single target variable using combining rules. This makes it possible to succinctly parameterize a potentially large Bayesian network where the nodes might have a large number of parents.

Consider the ER diagram in Figure 5. Entity types such as *Student* and *Course* are shown in rectangular boxes. Entities are objects that exist in the world [18]. Each entity has an attribute that describes the entity. The attributes are shown in ovals. *Course* has an attribute *Diff* (which represents course difficulty) and *Student* has an attribute *Sat* (which represents his/her satisfaction level). Note that we have omitted other attributes like *Id*, *Name* etc for brevity. *Takes* is the relationship between the entities *Course* and *Student* and is represented as a diamond in the Figure 5. Here *Takes* is a many-

Name
John
Mary
Tom

Student Table

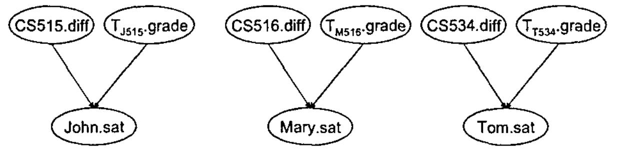
Student	Course
John	CS515
Mary	CS516
Tom	CS534

Takes Table

Course
CS515
CS516
CS534

Course Table

a. A Database for the Student-Course ER Model



b. Ground Bayesian network for the instantiated database

Figure 7: (a) A Database corresponding to the student ER model. (b) "Unrolled" ground Bayesian Network that is obtained by instantiating the variables of the FOCI statements with the database.

to-many relationship, as a student can take many courses and a course can be taken by many students. The relationship has an attribute *Grade* which indicates the grade of a student in a particular course.

In Figure 5, we can write

```
If {Student(s), Course(c), Takes(t,s,c)} then
    c.diff, t.grade Qinf s.satisfaction.
```

which means that the satisfaction of a student depends on the difficulty of the course that he took and the grade he obtained in that course. This corresponds to annotating the ER diagram with hyperarcs as shown in Figure 6.

This FOCI-annotated ER diagram can be "unrolled" to obtain an equivalent set of bayesian networks. To illustrate the point, consider Figure 7(a) which shows a part of the database (the entire database is not presented). There are three *Courses* and three *Students*. The relationship *Takes* contains a tuple for each *Student-Course* pair.[1] Figure 7(b) shows the ground bayesian network corresponding to the FOCI statement instantiated with values from the database. The instantiation of the *Takes* relationship is represented as $T_{studentcourse}$. For instance T_{J515} represents the *takes* relationship between *John* and *CS*515. As can be seen, the instantiated values of the influents and the resultants are the random variables in the ground bayesian network.

This example provides an insight into the semantics of the language. We could imagine each attribute of each object in the entire database to be a node of a giant bayesian network and the FOCI statements specify the arcs in the bayesian network i.e., the parents of a particular node is determined by the conditions of the FOCI statement. In this case, the condition is used to select the correct *student* − *course* pair from the database. In our language, we restrict the conditions to conjunctions of predicates as our primary goal is not to design the most expressive language but to design the most

[1]Note that each student takes only one course in this example. We can also handle cases in which a student can take more than one course.

useful and tractable one. Each *satisfaction* node in the Bayesian network will have a *conditional probability distribution*, $P(sat \mid c.\text{diff}, t.\text{grade})$.

FOCI makes it easier to convert hand-crafted domain knowledge into a representation suitable for learning. Given a fixed domain of objects and a database of facts about those objects, FOCI statements define Bayesian network fragments over the object attributes. FOCI statements capture the same kind of influence knowledge as in PRMs [22], BLPs [28], and DAPER models [25], with some differences. Unlike PRMs, which only allow path expressions, the conditions in FOCI statements can express arbitrary conjunctions of literals. In BLPs, the conditions are not syntactically separated from the influents. DAPER models attach arbitrary first-order conditions to the Bayes net arcs. FOCI not only restricts the conditions to conjunctions of literals, but also attaches them to hyper-arcs. We have used FOCI statements to express several pieces of useful domain knowledge in the Task-Based User Interface domain, the West Nile Virus domain, and an additional domain, the Grasshopper domain. The language has been found to be sufficiently powerful to capture the relevant domain knowledge in these domains. FOCI can also succinctly express most of the examples in a variety of domains considered in the statistical relational learning literature.

With relational probabilistic models, it is possible to describe a generalized conditional probability distribution in which a particular random variable (the "target") is conditioned on a set of parent variables in such a way that when the model is converted to ground form ("unrolled"), the number of parent nodes varies from one instance of the target variable to another. For example, the size of a population of mosquitos depends on the temperature and the rainfall each day since the last freeze. In one location, there might have been 19 days since the last freeze whereas in another location, there might have been only 3 days (see Figure 8(a)).

There are two main approaches to deal with this "multiple-parent" problem: aggregators and combining rules. An aggregator is a function that takes the *values* of the parent variables and combines them to produce a single aggregate value which then becomes the parent of the target variable. In the mosquito problem, we might define the total temperature and the total rainfall as aggregate variables. These are well defined for any number of parents, and they can be computed deterministically (shown as dashed lines in Figure 8(b)). The population node then has only two parents: TotalTemp and TotalRain.

The second approach to the multiple-parent problem is to have each parent or a small set of related parents produce a different child variable, and then combine the different child variable values using a deterministic or stochastic aggregating function. In the above example, the mosquito population of each day may be made a random function of a single temperature-rain pair and the populations from each day may be combined into a single value through a deterministic (e.g., max, min) or a stochastic (e.g., random choice) aggregating function (Figure 8(c)). The advantage of this method is that it can capture interactions between the Temp and Rain variables that are lost when temperature and rain are aggregated separately. In effect, the different days are "voting" about the probability of the mosquito population. This approach is formalized using "decomposable combining rules," e.g., noisy-Max, noisy-Min, or mean, which express a conditional probability distribution such as P(Pop|Temp1, Rain1, Temp2, Rain2, ..., Tempk, Raink) as a function of simpler distributions, P(Pop|Temp1, Rain1), P(Pop|Temp2,Rain2), etc. [36, 27].

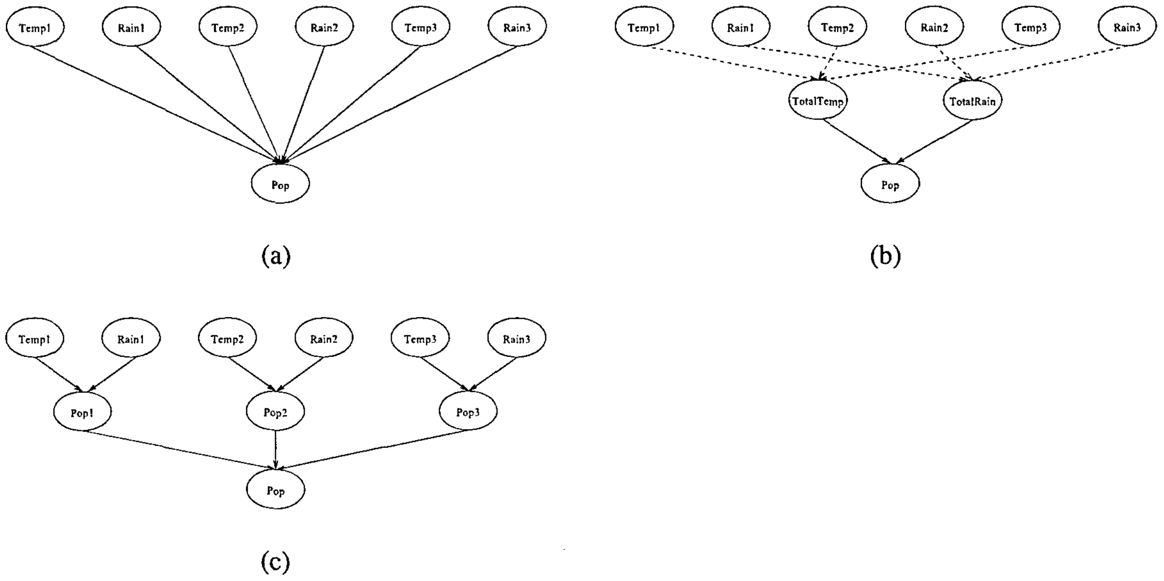


Figure 8: Three Bayesian networks describing the influence of daily temperature and rainfall on the population of mosquitos. (a) a network with no aggregation or combination rules leads to a very complex conditional probability distribution, (b) a network with separate aggregation for temperature and rainfall, (c) a network with separate prediction of the mosquito population each day followed by a combining rule to predict the overall population.

As a specific example of using combining rules in the task-based user interface domain, consider an intelligent desktop assistant that must predict the folder of a document. Assume that there are several tasks that a user can work on, such as proposals, courses, budgets, etc. The following FOCI statement says that a task and the role the document plays in that task influence its folder.

`If {task(t), document(d), role(d,r,t)` then t.id,r.id Qinf d.folder.}

Typically a document plays several roles in several tasks. For example, it may be the main document of one task but only a reference in some other task. Thus there are multiple task-role pairs $(t_1,r_1),\ldots,(t_m,r_m)$, each yielding a distinct folder distribution $P(d.folder \mid t_i.id, r_i.id)$. We need to combine these distributions into a single distribution for the folder variable. We could apply some kind of aggregator (e.g., the most frequently-occurring task-role pair) as in PRMs [22]. However, it is easy to imagine cases in which a document is accessed with low frequency across many different tasks, but these individual accesses, when summed together, predict that the document is stored in a convenient top-level folder rather than in the folder of the most frequent single task-role pair. This kind of summing of evidence can be implemented by a combining rule.

In the above example, a combining rule is applied to combine the distributions due to different influent instances of a single FOCI statement. In addition, combining rules can be employed to combine distributions arising from multiple FOCI statements with the same resultant. The statement in Figure 9 captures such a case (see Figure 10 for the unrolled network). The expression includes two FOCI statements. One statement is the task-role influence statement discussed above. The other statement

```
WeightedMean{
  If {task(t), doc(d), role(d,r,t)} then t.id, r.id Qinf (Mean) d.folder.
  If {doc(s), doc(d), source(s,d)} then s.folder Qinf (Mean) d.folder.
}
```

Figure 9: Example of specifying combining rules in FOCI

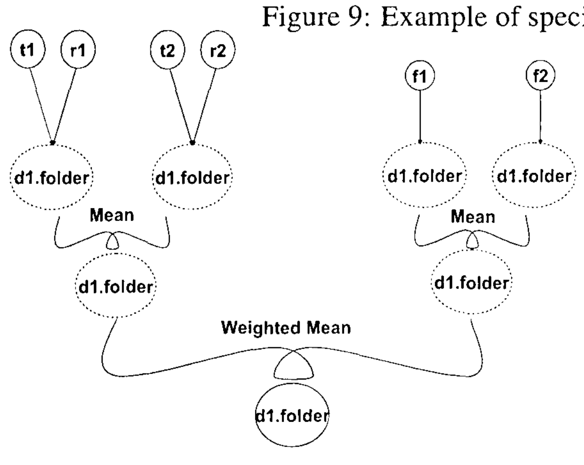


Figure 10: Use of Combining Rules to combine the influences of task and role on the one hand and the source folder on the other on the folder of the current document.

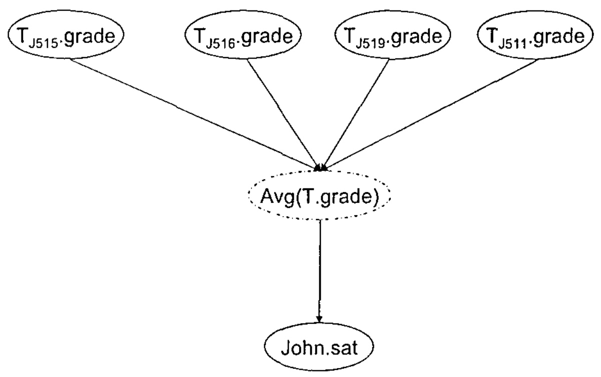


Figure 11: An "unrolled" Bayesian Network with aggregators. The grades in different courses are aggregated using the *Average* aggregator.

says that the folder of the source document of *d* influences *d*'s folder. By the "source document", we mean the document that was edited to create the current document. There can be multiple sources for a document. The distributions corresponding to different instances of the influents in the same statement are combined via the mean combining rule (indicated by the keyword "Mean"). The two resulting distributions are then combined with a weighted mean combining rule.

The other main approach to handling variable numbers of parents—aggregators—has its origin in database theory [22]. Aggregators are functions that take the *values* of parent variables and combine them to produce a single aggregate value. This value then becomes the parent of the *resultant* variable. In the student satisfaction example, we could say that the average grade that the student obtains in all the courses would influence his satisfaction. This would correspond to a FOCI statement,

```
If {Student(s), Course(c), Takes(t,s,c)} then
        Avg([ t.score | s ]) Qinf s.satisfaction.
```

Consider the database from Figure 7. Suppose *John* takes 4 courses {*CS515, CS516, CS511, CS519*}. The ground bayesian network when the above FOCI statement with aggregation is instantiated with *John* is shown in Figure 11. It can be observed that the grades that *John* has received in these courses are averaged and this is shown as a dotted node in the network. This would correspond to a hidden node in the network. This aggregated node serves as a parent for the *satisfaction* node and each *satisfaction* node will have a conditional probability distribution $P(Sat|AverageGrade)$ associated with it.

In the above statement, we are fixing the student and then averaging over the grades of all the courses that the student took. Instead we may need to average over the courses. For example, we may define the *meangrade* of a course as the average of the grades of all the student. The FOCI statement to define this is,

```
If {Student(s), Course(c), Takes(t,s,c)} then
                         c.meangrade := Avg([ t.score | c ])
```

which says that we aggregate over all the grades of a course c. We use the | notation as a shorthand for set formers for the ease of specifying the statements. This notation is used to specify that while aggregating the instances of t, we would initially fix the course. Also, this allows us to move all the declarations inside the condition.

After designing FOCI, we formulated learning algorithms based on gradient descent to learn parameters of the FOCI statements from supervised training data. One of the main issues here is that there are many hidden variables that correspond to the inputs of the combining rules. The algorithms learn the hidden parameters by minimizing the mean squared error over the training set or by maximizing the log likelihood of the training data through gradient descent. We obtained encouraging results in our experiments on synthetic data generated to test the algorithms. In addition, we have developed and implemented an EM algorithm for learning parameters with the 'weighted average' combining rule. The EM algorithm starts with some initial values for the FOCI-statement parameters. It then iteratively estimates the probability that the final label of the target attribute of the example came from each FOCI statement. These probabilities are treated as fractional counts of the target attribute for each FOCI-statement, and are in turn used to compute the maximum likelihood estimates of the FOCI-statement parameters. The EM algorithm is tested on some synthetic and real data sets and is found to perform as well as the gradient-descent algorithms.

3.1.2 Learning FOCI Model Parameters

We now present algorithms for learning the parameters of the combining rules and the conditional probability tables (CPTs). For additional details please see [35]. Consider a generic influence statement S_i:

$$\text{If } \langle condition \rangle \text{ then } X_1^i, \ldots, X_k^i \overset{Qinf}{\succ} Y.$$

We assume without loss of generality that each influence statement S_i ('rule i' for short) has k influents, X_1^i through X_k^i (which we jointly denote as $\mathbf{X}^i$), that influence the target variable. When this rule is instantiated or "unrolled" on a specific database, it generates multiple, say m_i, sets of influent instances, which we denote as $\mathbf{X}_1^i \ldots \mathbf{X}_{m_i}^i$. This is shown in Figure 12. In the figure, the instantiations of a particular statement are combined with the *mean* combining rule, while the distributions of the different statements are combined via the *weighted mean* combining rule. The role of the combining rule is to express the probability $P_i(Y|\mathbf{X}_1^i \ldots \mathbf{X}_{m_i}^i)$ as a function of the probabilities $P_i(Y|\mathbf{X}_j^i)$, one for each j, where P_i is the CPT associated with S_i. Since these instance tuples are unordered and can be arbitrary

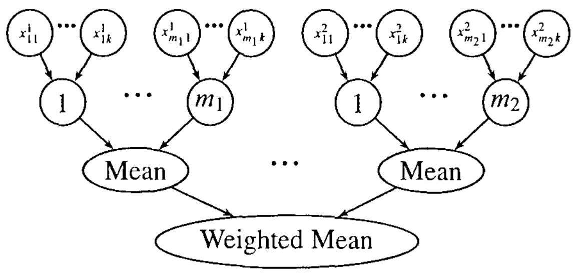


Figure 12: Unrolling of FOCI statements

in number, our combining rule should be symmetric. For example, with the mean combining rule, we obtain:

$$P(y|\mathbf{X}_1^i \dots \mathbf{X}_{m_i}^i) = \frac{1}{m_i} \sum_{j=1}^{m_i} P_i(y|\mathbf{X}_j^i) \tag{1}$$

If there are r such rules, we need to estimate the conditional probability $P(Y|X_{1,1}^1 ... X_{m_r,k}^r)$. Since each rule is distinctly labeled and its instances can be identified, the combining rule need not be symmetric, e.g., weighted mean. If w_i represents the weight of the combining rule, the "weighted mean" is defined as:

$$P(Y|X_{1,1}^1 ... X_{m_r,k}^r) = \frac{\sum_{i=1}^r w_i P(Y|\mathbf{X}_1^i \dots \mathbf{X}_{m_i}^i)}{\sum_{i=1}^r w_i} \tag{2}$$

We write $x_{j,1}^i, \dots, x_{j,k}^i \equiv \mathbf{x}_j^i$ to denote the values of $\mathbf{X}_j^i$ and y to denote the value of Y. We write $\theta_{y|\mathbf{x}^i}$ to denote $P_i(y|\mathbf{x}^i)$. Note that in this case we omit the subscript j because the parameters θ do not depend on it.

We now derive the gradient-descent algorithm for the mean-squared error function for the prediction of the target variable, when multiple FOCI-statements are present. Let the l^{th} training example e_l be denoted by $((x_{l,1,1}^1, \dots, x_{l,m_{l,r_l},k}^{r_l}), y_l)$, where $x_{l,j,p}^i$ is the p^{th} input value of the j^{th} instance of the i^{th} rule on the l^{th} example. The predicted probability of class y on e_l is given by

$$P(y|e_l) = \frac{1}{\sum_i w_i} \sum_i^{r_l} \frac{w_i}{m_{l,i}} \sum_j^{m_{l,i}} P_i(y|\mathbf{x}_{l,j}^i). \tag{3}$$

In the above equation, r_l is the number of rules the example satisfies, i is an index of the applicable rule, and $m_{l,i}$ is the number of instances of rule i on the l^{th} example. The squared error is given by

$$E = \frac{1}{2} \sum_{l=1}^{n} \sum_y (I(y_l, y) - P(y|e_l))^2. \tag{4}$$

Here y is a class label, and y_l is the true label of l^{th} example. $I(y_l, y)$ is an indicator variable that is 1 if $y_l = y$ and 0 otherwise. Taking the derivative of negative squared error with respect to $P(y|\mathbf{x}^i) = \theta_{y|\mathbf{x}^i}$, we get

$$\frac{-\partial E}{\partial \theta_{y|\mathbf{x}^i}} = \sum_{l=1}^{n} \sum_{y} \Bigg[(I(y_l,y) - P(y|e_l)) \left[\frac{1}{\sum_{i'} w_{i'}} \frac{w_i}{m_{l,i}} \#(\mathbf{x}^i|e_l) \right] \Bigg]. \tag{5}$$

Here $\#(\mathbf{x}^i|e_l)$ represents the number of occurrences of the tuple $\mathbf{x}^i$ in the x-instances of the i^{th} rule of example e_l. Gradient descent increments each parameter $\theta_{y|\mathbf{x}^i}$ by - $\alpha\frac{\partial E}{\partial \theta_{y|\mathbf{x}^i}}$ in each iteration. After each iteration, the parameters are normalized so that the distributions are well-defined[2].

When we adjust the weights of the rules using examples, we preserve the sum of the weights of the matching rules in each example, so that the overall sum of all weights is preserved, and the dependencies between the weights are properly taken into account when we take the derivatives. In particular, the gradients with respect to rule weights are computed as follows:

$$\frac{-\partial E}{\partial w_i} = \sum_{l=1}^{n} \left[\delta(e_l,i) - \frac{1}{r_l} \sum_{r} \delta(e_l,r) \right], \tag{6}$$

where $\delta(e_l,r)$ is given by

$$\sum_{y} (I(y_l,y) - P(y|e_l)) \frac{1}{\sum_{i'} w_{i'}} \frac{1}{m_{l,r}} \sum_{j} P_r(y|\mathbf{x}^r_{l,j}) \tag{7}$$

In the context of probabilistic modeling, it is more common to maximize the log likelihood of the data given the hypothesis [6]. From the definition of $P(y_l|e_l)$, we can see that this is

$$L = \sum_{l} \log P(y_l|e_l). \tag{8}$$

Taking the derivative of L with respect to $P(y|\mathbf{x}^i) = \theta_{y|\mathbf{x}^i}$, gives

$$\frac{\partial L}{\partial \theta_{y|\mathbf{x}^i}} = \sum_{l} \frac{1}{P(y_l|e_l)} \frac{1}{\sum_{i'} w_{i'}} \sum_{i}^{r_l} \frac{w_i}{m_{l,i}} \#(\mathbf{x}^i|e_l). \tag{9}$$

As before, the partial derivative of L with respect to the weights is given by

$$\frac{\partial L}{\partial w_i} = \sum_{l=1}^{n} \left[\delta(e_l,i) - \frac{1}{r_l} \sum_{r} \delta(e_l,r) \right]. \tag{10}$$

But now,

$$\delta(e_l,r) = \frac{1}{P_r(y_l|e_l)} \frac{1}{\sum_i w_i} \frac{1}{m_{l,r}} \sum_{j} P_r(y_l|\mathbf{x}^r_{l,j}). \tag{11}$$

[2]Though we did not find it necessary in this problem, another approach would be to reparameterize the objective function with exponentials, thus incorporating the normalization constraint.

We have found that it is important to have separate learning rates for the CPT parameters and for the combining-rule weights. In particular, the weights should be updated much more slowly than the conditional probabilities. This is because the each iteration of each example only changes a few of the CPT parameters, whereas it changes most of the weights.

Expectation-Maximization (EM) is a popular method to compute maximum likelihood estimates given incomplete data [14]. EM iteratively performs two steps: the *Expectation* step, where the algorithm computes the expected values of the missing data based on the current parameters, and the *Maximization* step, where the maximum likelihood of the parameters is computed based on the current expected values of the data. We adapted the EM algorithm for two-component mixture models from

Table 1: EM Algorithm for parameter learning in FOCI

1. Take initial guesses for parameters θ and weights w_i

2. E Step: $\forall i$ and for each instantiation of each rule, compute the responsibilities

$$\gamma^i_{l,j} = \frac{(w_i)\frac{1}{m_{l,i}}\theta_{y|\mathbf{x}^i_{l,j}}}{\sum_{l,i',j}(w_{i'})\frac{1}{m_{l,i'}}\theta_{y|\mathbf{x}^{i'}_{l,j}}}$$

3. M Step: Compute the new parameters:

$$\forall i, \mathbf{x}^i \quad \theta_{y|\mathbf{x}^i} = \frac{\sum_{l,j}\gamma^i_{l,j} I_{y,\mathbf{x}^i}}{\sum_{y,l,j}\gamma^i_{l,j} I_{y,\mathbf{x}^i}}$$

and, if instantiations of at least two rules are present in l, compute

$$\forall i \quad w_i = \left(\sum_{l,j}\gamma^i_{l,j}\right)/n_2$$

where n_2 is the number of examples with two or more rules instantiated.

4. Continue E and M steps until convergence.

[24]. Consider n rules with the same resultant. Accordingly, there will be n distributions that need to be combined via a weighted mean. Let w_i be the weight for rule i, such that $\sum_i w_i = 1$.

The EM algorithm for parameter learning in FOCI is presented in Table1. In the *expectation* step, we compute the responsibility of each instantiation of each rule. The responsibilities reflect the relative density of the training points under each rule [24]. Note that we consider the weight of the current rule and the number of instantiations of the current rule while computing the responsibility of an instantiation of the current rule. In the *maximization* step, we use these responsibilities to update the CPTs. We use the responsibilities of the instantiations of an example to compute the weights if at least two rules are instantiated in the example. If an example matches less than two rules, the weights do not affect the distribution.

For instance, consider the update of $P(y_1 \mid \mathbf{x}^i)$. This is the fraction of the sum of all the responsibilities when $Y = y_1$ over all Y given $\mathbf{x}^i$. Likewise, the weight of the current rule is the fraction of the sum of the responsibilities of all instantiations of the rule over the number of examples with two or more rules instantiated.

3.1.3 Experimental Evaluation

In this section, we describe results on the two data sets that we employed to test the learning algorithms. The first is based on the folder prediction task, where we applied two rules to predict the folder of a document. The second data set is a synthetic one that permits us to test how well the learned distribution matches the true distribution. We present the results for both the experiments and compare them with the propositional classifiers.

Folder Prediction in a Task-Based User Interface Domain

We employed the two rules that were presented earlier in Figure 9, combined using the weighted mean. As part of the Task Tracer project [17], *i.e.,* task-based user interface domain, we collected data for 500 documents and 6 tasks. The documents were stored in 11 different folders. Each document was manually assigned to a role with respect to each task with which it was associated. A document was assigned the *main* role if it was modified as part of the task. Otherwise, the document was assigned the *reference* role, since it was opened but not edited. A document is a *source* document if it was opened, edited, and then saved to create a new document or if large parts of it were copied and pasted into the new document. Since the documents could play several roles in several tasks, the number of $\langle t, r \rangle$ pairs vary[3].

We applied Gradient Descent and EM algorithms to learn both the parameters of the CPTs and the weights of the weighted mean combining rule. We employed 10-fold cross-validation to evaluate the results. Within each fold, the learned network was applied to rank the folders of the current document and the position of the correct folder in this ranking was computed (counting from 1). The results are shown in Table 2, where the counts report the total number of times (out of 500) that the correct folder was ranked 1st, 2nd, etc. The final row of the table reports the mean reciprocal rank of the correct folder (the average of the reciprocals of the ranks). It is clear from the table that all the three relational algorithms performed very well: almost 90% of the documents had their correct folders ranked as 1 or 2 by all three algorithms[4].

To compare these results with propositional learners, we flattened the data using as features the numbers of times each task-role pair and each source folder appears in each example. We then used Weka to run J48 and Naive Bayes algorithms on this new dataset. J48 on the flattened data also performs as well as the relational classifiers while Naive Bayes does a little worse on the same data

[3]On average, each document participated in 2 $\langle t, r \rangle$ pairs, although a few documents participated in 5 to 6 $\langle t, r \rangle$ pairs.

[4]If the algorithm were to rank the folders at random, the score would be around 0.2745.

set. All the relational algorithms attributed high weights to the second rule compared to the first (see Table 3).

Rank	EM	GD-MS	GD-LL	J48	NB
1	349	354	346	351	326
2	107	98	113	100	110
3	22	26	18	28	34
4	15	12	15	6	19
5	6	4	4	6	4
6	0	0	3	0	0
7	1	4	1	2	0
8	0	2	0	0	1
9	0	0	0	6	1
10	0	0	0	0	0
11	0	0	0	0	5
Score	0.8299	0.8325	0.8274	0.8279	0.797

Table 2: Results of the learning algorithms on the folder prediction task. GD-MS: Gradient descent for Mean Square error; GD-LL: Gradient descent for log-likelihood; J48: Decision Tree; NB: Naive Bayes for loglikelihood.

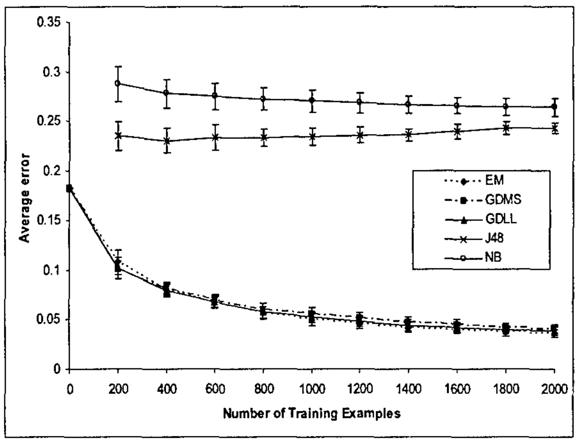
To test the importance of learning the weights, we altered the data set so that the folder names of all the sources were randomly chosen. As can be seen in Table 3, with this change, the source document rule is assigned low weight by the learning algorithms with a small loss in the score.

		EM	GD-MS	GD-LL
Original data set	Weights	$\langle .15, .85 \rangle$	$\langle .22, .78 \rangle$	$\langle .05, .95 \rangle$
	Score	.8299	.8325	.8274
Modified data set	Weights	$\langle .9, .1 \rangle$	$\langle .84, .16 \rangle$	$\langle 1, 0 \rangle$
	Score	.7934	.8021	.7939

Table 3: Results of learning the weights in the original data set and the modified data set.

Synthetic Data Set

To realistically model a complex real-world domain, it is not enough to have a good classification accuracy on a single task. To use these predictions in complex inferences, it is important to accurately model the probability distributions. To estimate the accuracy of the learned model, we constructed a synthetic data set. The data are generated using a synthetic target as defined by two FOCI statements, each of which has two influents and the same target attribute. The two influents in each rule have a range of 10 and 3 values respectively. The target attribute can take 3 values. The probability values in the distribution of the synthetic target are randomly generated to be either between 0.9 and 1.0



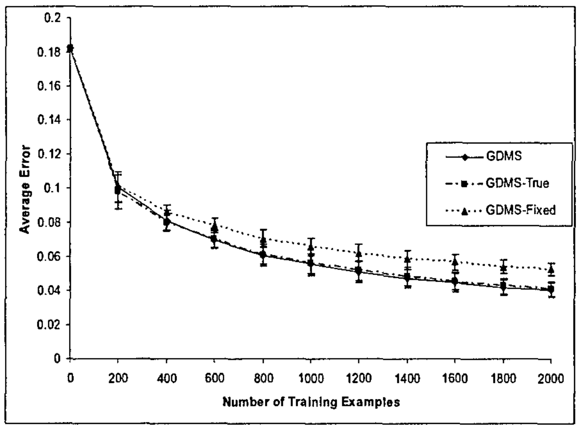


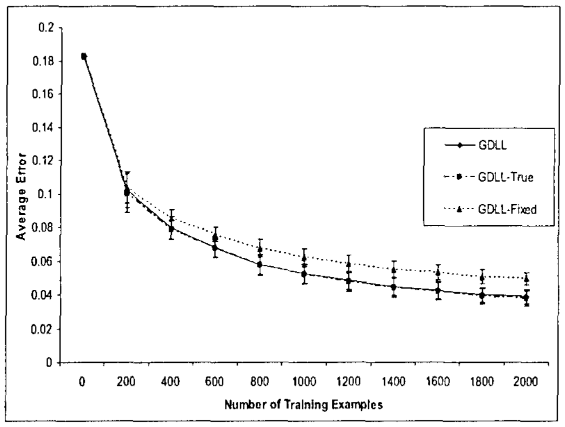
Figure 13: Learning curves for the synthetic data. EM: Expectation Maximization; GDMS: Gradient descent for Mean Square error; GDLL: Gradient descent for log likelihood; J48: Decision tree; NB: Naive Bayes.

Figure 14: Learning curves for mean squared gradient descent on the synthetic data. GDMS: learning the weights; GDMS-True: gradient descent with true weights; GDMS-Fixed: gradient descent with weights fixed as $\langle 0.5, 0.5 \rangle$.

or between 0.0 and 0.1. This is to make sure that the probabilistic predictions on examples are not too uncertain. The rule weights are fixed to be 0.1 and 0.9 to make them far from the default, 0.5. Each example matches a rule with probability 0.5, and when it does match, it generates a number of instances randomly chosen between 3 and 10. This makes it imperative that the learning algorithm does a good job of inferring the hidden distributions both at the instance level and the rule level.

We trained the learning algorithms on 30 sets of 2000 training examples and tested them on a set of 1000 test examples. The average absolute difference between corresponding entries in the true distribution and the predicted distribution was averaged over all the test examples. Like the folder data set, we flattened the data set by using the counts of the instances of the parents as features and used Weka to run J48 and Naive Bayes on this modified data set. The results are presented in Figure 13. All three relational algorithms have a very low average absolute error between the true and the predicted distribution. The overlapping of the error bars suggests that there is no statistically significant difference between the algorithms' performances. On the other hand, the propositional classifiers perform poorly on this measure compared to the relational algorithms.

As with the folder data set, we wanted to understand the importance of learning the weights. Hence, for each learning algorithm, we compared three settings. The first setting is the normal situation in which the algorithm learns the weights. In the second setting, the weights were fixed at $\langle 0.5, 0.5 \rangle$. In the third setting, the weights were fixed to be their true values. The results are presented in Figures 14, 15, and 16. There are three curves in each figure corresponding to the three settings. In all three algorithms, the first setting (weights are learned) gave significantly better error rates than the second setting (weights fixed at $\langle 0.5, 0.5 \rangle$) (Figures 14,15,16). This clearly demonstrates the importance of learning the weights. There was no significant difference between learning the weights and knowing



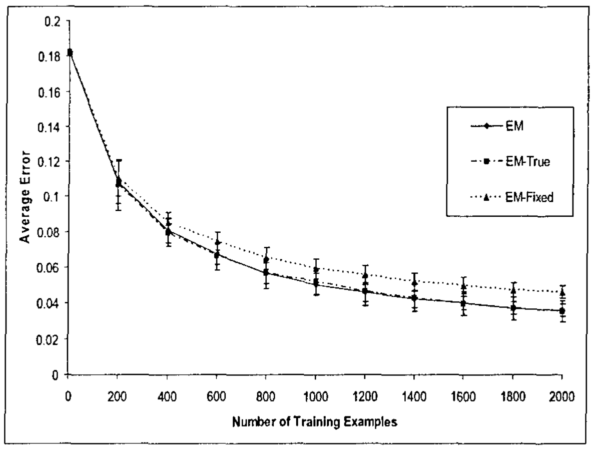


Figure 15: Learning curves for log-likelihood gradient descent on the synthetic data. GDLL: learning the weights; GDLL-True: gradient descent with true weights; GDLL-Fixed: gradient descent with weights fixed as $\langle 0.5, 0.5 \rangle$.

Figure 16: Learning curves for EM on the synthetic data. EM: learning the weights; EM-True: EM with true weights; EM-Fixed: EM with weights fixed as $\langle 0.5, 0.5 \rangle$.

the true weights. This shows that our algorithms effectively learn the weights of the combining rules.

3.2 Task-Based User Interface: Infrastructure and Domain Knowledge

In the domain of task-based user interface, we have developed TaskTracer - a software system designed to help highly multitasking knowledge workers to rapidly locate, discover, and reuse past processes they used to successfully complete tasks [17, 41, 3]. The system monitors users' interaction with a computer, collects detailed records of users' activities and resources accessed, associates (automatically or with users' assistance) each interaction event with a particular task, enables users to access records of past activities and quickly restore task contexts. Below we provide diagrams and examples to illustrate the domain knowledge base we have constructed for the task-based user interface domain. In addition, we give an overview of TaskTracer's infrastructure illustrating the novel Publisher-Subscriber architecture for collecting and processing users' activity data.

Figure 17 shows an ER Diagram describing the relationship between the user and the project the user is working on, as well as their relationships to tasks, documents, and files. Based on the dependencies in the ER Model, we constructed the corresponding PERMs. A couple of examples are shown in Figures 18 and 19. Note that corresponding FOCI statements (discussed in Section 3.1) used to represent the domain knowledge are also given. Figure 18 expresses the knowledge that the folder of a source document influences the folder of the document that used the former as a source. Figure 19 simply says that a pair of specific task and role together would tell us something about the sender of an email message.

Through an extensive data-collection framework, TaskTracer collects detailed observations of user

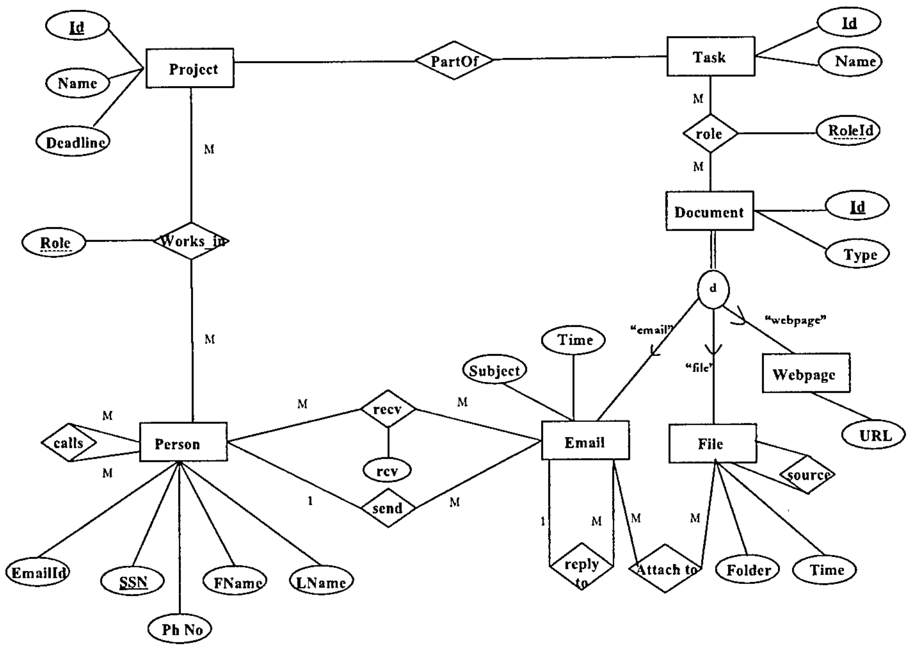


Figure 17: ER Model for TaskTracer

interactions in the common productivity applications used in knowledge work: email, word processing, spreadsheets, and Internet browsers. At the initial stage of data collection, users manually specify what tasks they are doing, so that each action of the user (UI event) is tagged with a particular task identifier. TaskTracer monitors Microsoft Office, Visual Studio, and Internet Explorer applications by installing .NET COM addin objects with the Extensibility and IObjectWithSite interfaces. These addins monitor

If {doc(d),doc(s)source(s,d)} then s.folder Qinf d.folder.

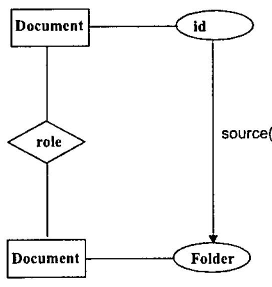
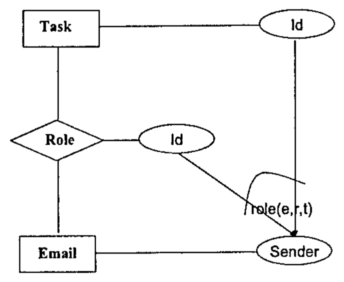


Figure 18: TaskTracer PERM Example 1

If {email(e),task(t) role(e,r,t)} then t.id,r.id Qinf e.sender.



Note: Here *sender* is a foreign key of the Email table that references the person id

Figure 19: TaskTracer PERM Example 2

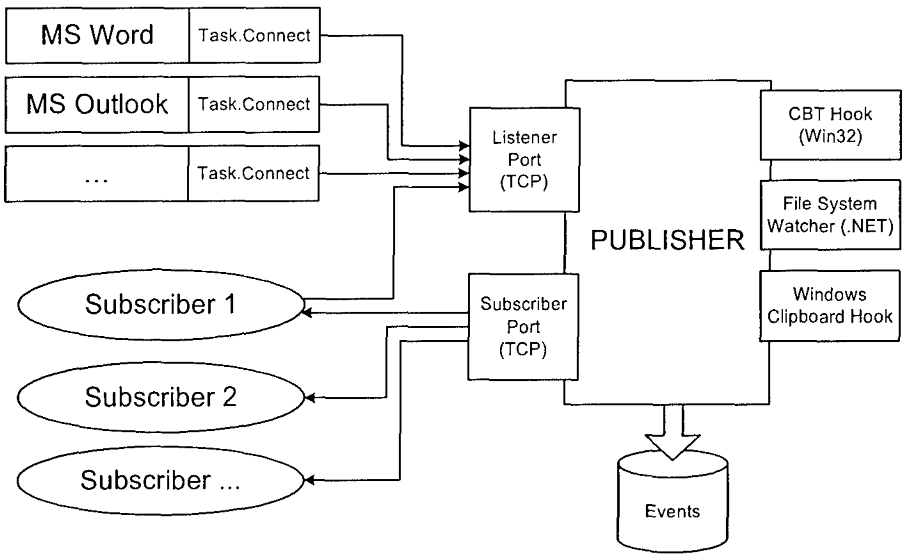


Figure 20: TaskTracer Publisher-Subscriber Architecture

Microsoft applications as soon as they are launched and are intimately bound to the applications. No user intervention or shell application is required once TaskTracer is installed. The addins also access the richer event sets of the Microsoft Office internal Visual Basic for Applications (VBA) compiler. Using VBA, TaskTracer can monitor a richer event set by working around Microsoft Office COM limitations. TaskTracer also monitors Windows at the operating system level with such events as window focus, clipboard and file creation events. To monitor applications and the operating system, TaskTracer compnents are written in C#, C++, and VB.NET.

TaskTracer separates the user interface and data analysis components from the event collection components by using a Publisher-Subscriber Architecture (Figure 20). The Publisher collects data about the user's activities and disseminates this event data to one or more Subscribers. Each Subscriber can process the event data from the Publisher in a different way. Some Subscribers not only receive EventMessages from the Subscriber Port but also send EventMessages to the Listener Port. For example, TaskExplorer (a UI for task switching described in [17]), sends TaskBegin messages to the Listener Port every time the user selects a new task. The Publisher-Subscriber architecture of TaskTracer has several powerful advantages over a monolithic approach. These advantages include the following:

- Data collection is separated from data analysis and the user interface.
- No prejudgments are made about the data schema of the data collected.
- Multiple researchers can work on multiple projects with multiple data schemas that subscribe to the Publisher without interfering with each other.

- Raw event data stored by the Publisher is like a tape-recorder and can be analyzed and even replayed at a later time.

The separation of the data collection components from the analysis and presentation components allows rapid application development of new software projects by researchers. For example, to create a new UI, a researcher only needs to understand the structure of the EventMessage and the use of the TCP protocol. The ability to send EventMessages simultaneously to many Subscribers allows separate independent research projects to assemble a suite of applications that can function as a single application.

We have also run several experiments using the data we collected from TaskTracer. We developed algorithms that could predict the folder the user is going to need (FolderPredictor [3]) and recognize user task and email messages (TaskPredictor [41]). In the folder prediction problem, a simple domain knowledge such as "a user would most likely need to work on recently-accessed folders" is incorporated into the algorithm in terms of *recency weighting* (see [3] for more details). The main idea behind FolderPredictor is that if we have observations of a user's previous file access behavior, we can recommend one or more folders directly to the user at the moment he/she needs to locate a file. These recommended folders are predictions - the result of running simple machine learning algorithms on the user's previously observed interactions with files.

From the raw event stream, the main TaskPredictor extracts a sequence of Window-Document Segments (WDSs). A WDS consists of a maximal contiguous segment of time in which a particular window has focus and the name of the document in that window does not change. In our approach, a new WDS is defined to begin when one of the following events happens:

- *Navigate* (Internet Explorer): the browser displays a new webpage;
- *OsWindowFocus* (all applications): a different window gains the focus;
- *Open* (MS Office): the user opens a new file
- *SaveAs* (MS Office): the user saves a file under a new name;
- *New* (MS Office): the user creates a new blank document.

TaskPredictor.WDS attempts to make a prediction for each WDS. To do this, it extracts the following information from each WDS: the window title, the file pathname, and (for web pages) the website URL. It heuristically segments these into a set of "words" and then creates a binary variable x_j in the feature set for each unique word. If this word appears in the event, x_j is 1, otherwise x_j is 0.

TaskPredictor.email does not use the WDS event stream. Instead, it attempts to make a prediction for each incoming email message. It creates a boolean feature for each observed email sender (the "FROM" field), one boolean feature for each observed *set* of email recipients (the union of the "FROM", "TO", "CC", and "BCC" fields), and one boolean feature for each distinct word observed in the "SUBJECT" field. Note that each set of recipients is treated as a separate feature, so an email message sent to $\{A,B\}$ might have no (true) features in common with an email message sent to $\{A,B,C\}$ unless they were from the same person or had the same words in the subject. We did not find that the words in the email body had any additional predictive value.

We have employed three methods to build TaskPredictor.

1. *Classification Thresholds.* Using a threshold θ, predictions are only made when the posterior probability exceeds θ. If the cost of prediction error is α, the cost of making no prediction is β, and the cost of a correct prediction is 0, then the threshold can be set to $\theta = 1 - \frac{\beta}{\alpha}$.

2. *Feature Selection using Information Gain.* Mutual information is used to select and reduce the number of features which in turn reduces the complexity of the learned hypothesis. Our experiments show that when the number of selected features is about 200, learning and prediction is most efficient.

3. *Hybrid Naive Bayes+SVM classifier.* The Naive Bayes classifier is used to estimate the posterior probability of the feature vector x. A decision to predict is only made whenever $P(x) > \theta$. After this decision has been made, the SVM is used to predict the task. Experimental results show that the hybrid method gives performance equal to or better than the best single method (Naive Bayes or SVMs).

3.3 Domain Knowledge for West Nile Virus and Grasshopper Domains

In order to gain a deeper understanding of the problems we are faced with and as part of our technical analysis, we have constructed domain models for both Grasshopper infestation and the spread of the West Nile Virus. Data for the former was obtained from the Oregon Department of Agriculture and Oregon Climate Service. We obtained data for the West Nile Virus through an NSF-sponsored project. Figure 21 shows the UML diagram for the West Nile Virus domain. The diagram shows the entities as well as the relationships between these entities that are necessary to model the spread of the virus. Mosquitos from a pool could bite birds, horses, and people. Birds, horses, and people are in turn associated with the specific location of their habitat.

Figure 22 shows the UML diagram for the Grasshopper domain. Grasshoppers go through three phases of development: egg, nymph (immature), and adult. Since the development of a grasshopper from egg to adult is largely affected by the temperature and precipitation, which in turn vary over the entire year, the UML diagram includes daily temperature ranges `(Tmin,Tmax)` and precipitation levels. Figure 23 shows a probabilistic entity relationship model (PERM) for the the Grasshopper domain. The model shows in detail the relationships and dependencies between variables. For instance, the `severity` of a `DrySpell` is affected by the maximum temperature `Tmax` of all of the days that occur `duringl` the time interval of the `DrySpell`. The severity of the dry spell affects, in turn, the number of eggs `Megg` laid by adult female grasshoppers. The number of eggs `Megg` deposited in the previous year affects the number of eggs `Negg` that will hatch into `Nymphs` in the year that follows. Variability in the temperature `Tmax` also affects the `severity` of a `ColdSpell`. The grasshopper eggs are laid in the ground, and their rate of development (and therefore, their hatching date) is determined by the daily temperature during the `EggPhase`. Once the eggs hatch into nymphs, the survival of the nymphs is determined by the severity of the `ColdSpells`.

Our analyses in these domains have given us opportunities in understanding better the kinds of domain knowledge that might be exploited for learning with very small training sets. For example, the prevalence of monotonicity relationships (among several others) in these domains such as that of grasshopper population and dry spells, and between egg hatching and cold spells have increased our

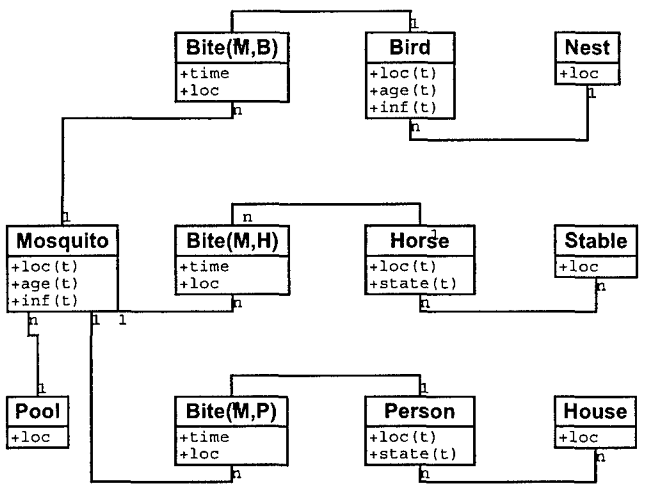


Figure 21: UML diagram for West Nile Virus

motivation in finding effective techniques of exploiting them. The language FOCI (Section 3.1) and the use of monotonicity constraints in Bayesian network parameter learning (Section 3.4) reflect such efforts.

3.4 Incorporating Constraints in Machine Learning Algorithms

Computer systems constructed with machine learning give the best known performance in many domains including speech recognition, optical character recognition, bio-informatics, biometrics, anomaly detection, and information extraction. However, it is not correct to view these systems as having been constructed purely from data. Rather, every practical machine learning system combines knowledge engineering with data. A critical and time-consuming part of building any successful machine learning application is the feature engineering, feature selection, and algorithm selection required to effectively incorporate domain knowledge. One of the reasons this process is so time consuming is that machine learning tools do not provide very many ways of expressing domain knowledge. In particular, there are many forms of prior knowledge that an expert might have that cannot be accepted or exploited by existing machine learning systems. This section discusses one particular form of prior knowledge—knowledge about qualitative monotonicities—and describes how this knowledge can be formalized and incorporated into learning algorithms for Bayesian networks.

Researchers in qualitative physics have developed several formal languages for representing qualitative influences [9, 30]. Others have shown that these qualitative influences could be usefully incorporated into learning algorithms, including the CN2 learning system, decision tree algorithms, and the back-propagation neural network algorithm [11, 2, 4, 13, 19, 12, 37, 26]. We have chosen Bayesian network learning algorithms because Bayesian networks already make it easy to express the causal structure and conditional independencies of a domain. We formalize qualitative monotonici-

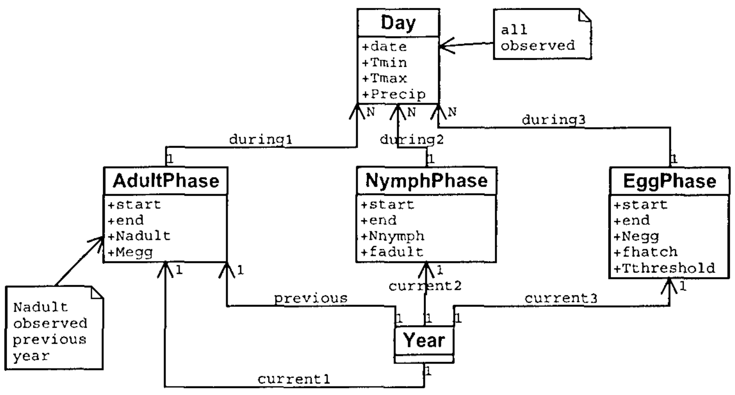


Figure 22: UML diagram for the Grasshopper domain

ties in terms of first-order stochastic dominance, building on the work of Wellman [45]. This in turn places inequality constraints on the Bayesian network parameters and leads naturally to an algorithm for finding the maximum likelihood values of the parameters subject to these constraints. We show experimentally that the additional constraint provided by qualitative monotonicities can improve the performance of Bayesian network classifiers, particularly on extremely small training sets.

3.4.1 Monotonicity Constraints

A monotonic influence, denoted $X \overset{Q+}{\succ} Y$ (or $X \overset{Q-}{\succ} Y$), informally means that higher values of X stochastically result in higher (lower) values of Y. For example, we might expect a greater risk for diabetes in persons with a higher body mass index.

Our basic question, then, is: how does the statement $X \overset{Q+}{\succ} Y$ constrain a probability distribution $P(Y \mid X)$? Although there are various definitions of stochastic ordering [32, 43, 44], we employ *first order stochastic dominance* (FSD) monotonicity, which is based on the intuition that increasing values of X shift the probability mass of Y upwards. This leads to the following three definitions.

Definition 1 (First Order Stochastic Dominance) *Given two probability distributions P_1 and P_2, and their respective cumulative distribution functions F_1 and F_2,*

$$P_1 \succeq_{(1)} P_2 \quad \mathit{iff} \quad \forall y\, F_1(y) \leq F_2(y). \tag{12}$$

Definition 2 (FSD Monotonicity) *We say Y is FSD isotonic (antitonic) in X in a context C if for all x_1, x_2 such that $x_1 \geq x_2$ (respectively, $x_1 \leq x_2$), we have*

$$P(Y \mid X = x_1, C) \succeq_{(1)} P(Y \mid X = x_2, C). \tag{13}$$

Definition 3 ($\overset{Q+}{\succ}$, $\overset{Q-}{\succ}$ Statements) *Suppose Y has multiple parents $X_1, X_2 \ldots X_q$. The statement X_i $\overset{Q+}{\succ}$ ($\overset{Q-}{\succ}$) Y means for all contexts (configurations of other parents) $C \in \times_{j \neq i} X_j$, that Y is FSD isotonic (antitonic) in X_i in context C.*

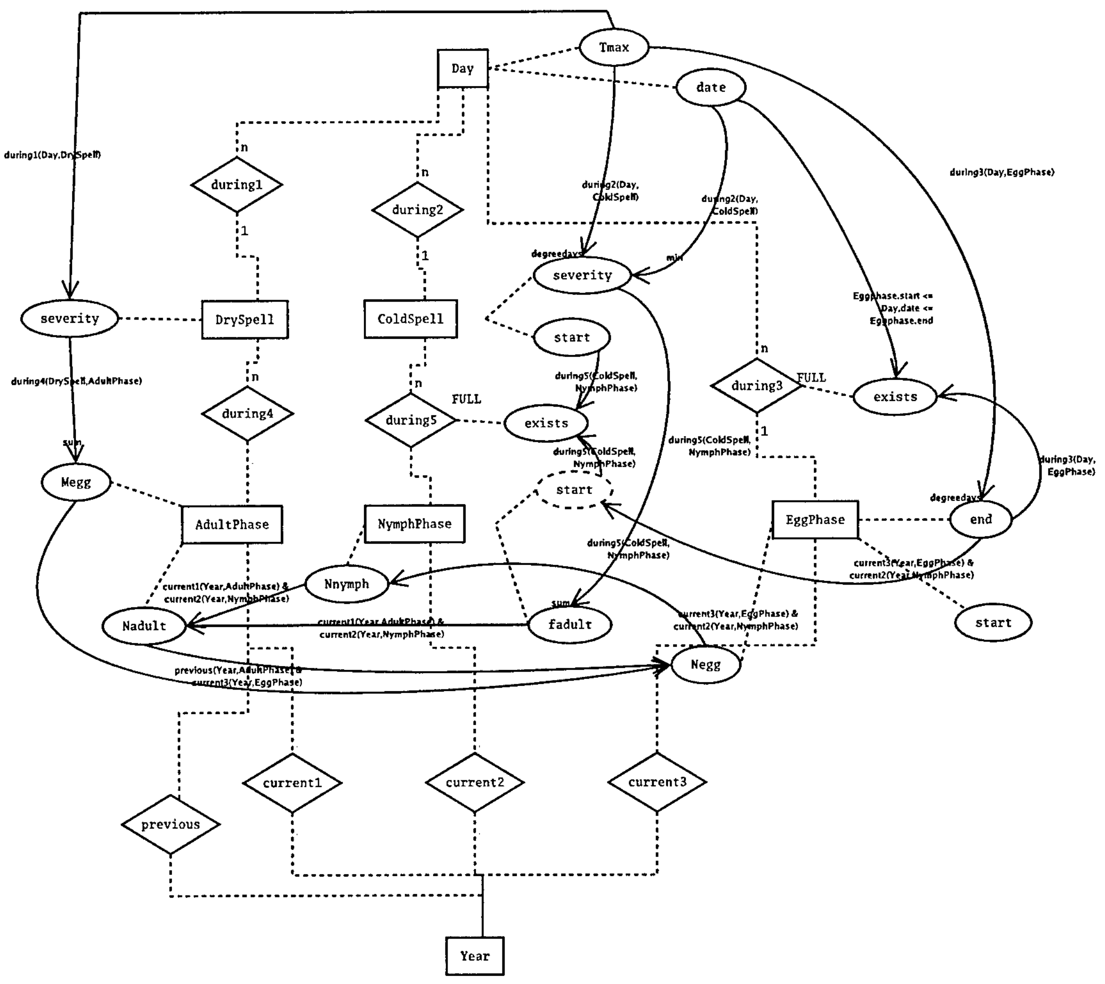


Figure 23: PERM for the Grasshopper domain

The last definition expresses a *ceteris paribus* assumption, namely that the effect of X on Y holds for *each* configuration of other parents of Y.[5] A simple example of the induced monotonic CPT constraints is shown in Figure 24.

Inference and Learning

Our approach to learning with prior knowledge of monotonicities is to apply the monotonicity constraints to the parameter estimation problem to find the set of parameter values that gives the best fit to the training data and prior while also satisfying the constraints. This is a form of constrained Maximum A Posteriori (MAP) estimation.

[5]Kuipers [30] discusses $M+$ monotonicity, which has an unconditional positive effect *across all* configurations of the other parents (i.e., the monotonicity effect is global and cannot be overridden by other influences). Other notations for $\overset{Q+}{\succ}$ include $S+$ (Wellman [45]) and $\propto_{Q+}$ (Forbus [20]).

	$P(Y \mid X)$		
x | y	0	1	2
0	θ_0	θ_3	θ_{03}
1	θ_1	θ_4	θ_{14}
2	θ_2	θ_5	θ_{25}

Constraints :
$\theta_0 \geq \theta_1$
$\theta_1 \geq \theta_2$
$\theta_0 + \theta_3 \geq \theta_1 + \theta_4$
$\theta_1 + \theta_4 \geq \theta_2 + \theta_5$

Figure 24: Example of a CPT for a three-valued variable Y given a three-valued parent X, with constraint $X \overset{Q+}{\succ} Y$. The values for θ_{ab} are given by $1 - \theta_a - \theta_b$.

Let G be the graph of a Bayesian network with nodes $X_1, \ldots, X_n$. Let θ_i denote the parameters of the CPT for X_i. Let θ_{ij} denote the row corresponding to parent configuration j, where $j \leq q_i$, the total number of parent configurations for X_i. Let θ_{ijk} denote the kth scalar in that vector, for $k \leq r_i$, the number of states of X_i. Finally, let θ denote the entire collection of parameters. The learning task involves finding the most probable values for θ given our fully observed data D and our prior ξ over the parameters. In this case, ξ is comprised of: (1) ξ^G, the conditional independence assumptions corresponding to the structure of G; (2) ξ^Q, the monotonicity constraints implied by our qualitative model Q; and (3) ξ^P, the prior over parameter values (e.g., a Dirichlet distribution for each conditional distribution). Under these priors, the likelihood for θ_i factors as

$$I_{(\theta_i \; satisf. \; \xi^Q)} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}}, \tag{14}$$

where $I_{(B)}$ is an indicator function that is 1 if B is true and 0 otherwise (in this case, if θ_i satisfies constraints ξ^Q), and N_{ijk} is the observed number of examples in which $X_i = k$ given parent configuration j, plus the appropriate parameter from a Dirichlet prior. This is a constrained optimization problem which we solve with an exterior penalty method, replacing the indicator function in Equation 14 with penalty functions that take on large negative values when the constraints are violated. This approach is prone to problems with convergence, but it is flexible and scales linearly with the number of constraints, which can be very large in our problems.

Inequality Constraint Margins

When observed data violates a monotonicity constraint, the maximum likelihood parameters are invariant to the parent configuration. It is debatable whether or not this is the intended or desired behavior. The proper solution would be a soft Bayesian prior on monotonicity which is updated with data, but for computational reasons we choose a simpler strategy. We enforce the strength of monotonicity by adding a margin to each inequality, replacing Equation (12) by

$$P_1 \succeq_{(1)} P_2 \quad \text{iff} \quad \forall y \, F_1(y) + \varepsilon \leq F_2(y). \tag{15}$$

We must be careful not to make ε too large, or it will strengthen the constraints to a point at which they have no solution (this is because inequalities are transitive, e.g.: $F_1(y) + \varepsilon \leq F_2(y), F_2(y) + \varepsilon \leq F_3(y), \ldots$). The maximum length of such a "chain" of inequalities is the Manhattan distance between the minimum-influence and maximum-influence corners of the CPT: $d_1^i = \left|\mathbf{pa}_i^{max} - \mathbf{pa}_i^{min}\right|_1 = \prod_{p\in\pi_i}(r_p - 1)$, where π_i is the set of parents of X_i. For example, if $q_i = 2$, and each parent has 3 states, we get $d_1^i = 4$ inequalities, and our maximum allowable value for ε is 0.25. Thus we define a global margin parameter ε and let each node X_i have its own ε_i margin, where $\varepsilon_i = \varepsilon/d_1^i$. Theoretically, ε could range up to 1.0, but we find that our current gradient search algorithms have difficulty finding the feasible region for ε greater than 0.2.

3.4.2 Experiments and Evaluation

To test the effectiveness of qualitative monotonicities, we conducted a series of experiments comparing Bayesian network classifiers learned with and without qualitative monotonicities. We have chosen five data sets from the UCI ML repository: auto-mpg[39], haberman[23], pima-indian-diabetes[42], breast-cancer-wisconsin[5], and car[10, 48]. For each of these data sets we constructed the network (KB structure) using domain knowledge and inserted monotonicity annotations ($\overset{Q+}{\succ}$ or $\overset{Q-}{\succ}$) on each of the network links according to our domain knowledge. This domain knowledge was based only on common knowledge (e.g., car purchasing) and information from previous publications concerning these data sets. In particular, we did not examine the data itself. We invested significant research in this task, and consider our constructed models the beginning of a benchmark corpus for monotonic learning algorithms.

We had originally chosen ten datasets, but of these, only five had a tractable Bayes net structure. The others had nodes with 8–11 incoming arcs, making the optimization task very difficult, and yielding low performance on all Bayes net classifiers (since we are relying on our domain knowledge, we chose to drop these datasets from this particular set of experiments rather than modify the networks in a way that disagreed with our causal understanding of the domain). We have, however, addressed this issue of learning from very sparse data using Bayesian networks with incoming arcs ranging from 8–11 in number. Constrained logistic regression seems a promising technique that addresses the problem (see Section 3.4.3). Also, we hypothesized that monotonicity constraints would prove more helpful at finer discretizations. To test this, for each data set, attributes with numeric values were discretized using Weka's (the Waikato Environment for Knowledge Analysis [46]) equal-frequency discretization tool to generate data sets with numbers of bins 2, 3, and 5, yielding a total of 15 data sets for our experiments. All class variables have two classes. Moreover, all incomplete rows in the data sets have been removed. To illustrate the point in this report, we will only show annotated knowledge bases and experimental results for two data sets. Additional details are given in [1].

Figure 25 shows the KB structure and monotonicity constraints for data set auto-mpg. In this data set, the classification problem is to predict whether a car has low (≤ 28) or high (> 28) mileage per gallon (mpg). auto-mpg has 392 instances of which 106 are labeled positive examples. Domain knowledge suggests that an increase in the number of cylinders (cylinders) usually leads to an increase in horsepower (horsepwr), displacement (disp), and vehicle weight (weight). An increase in weight leads to a decrease in mpg. The heavier the vehicle, the slower it accelerates (accel). The larger the displacement, the greater the horsepower, but the lower the mileage per gallon. Finally, newer models (modelyear) tend to be more fuel-efficient, as do vehicles imported from (origin) Japan (encoded as 1) as opposed to Europe or those produced in the United States (encoded as 0). These monotonicity

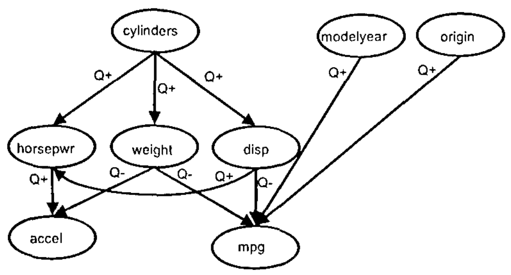


Figure 25: auto-mpg

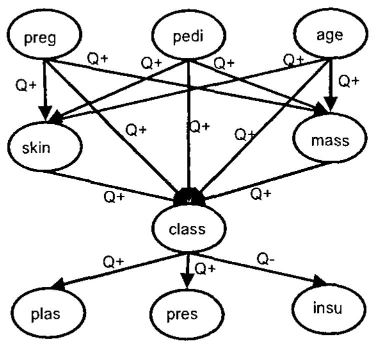


Figure 26: pima-indian-diabetes

1. Initialize the μ_{ijk} parameters at the unconstrained MLE point (found simply by counting the observations)
2. If this point satisfies the constraints, return it
3. Otherwise, initialize a weight w for the penalty functions
4. Take steps in the steepest direction of the penalized likelihood until convergence
5. If we converged outside the feasible region, increase the penalty weight and repeat the previous step.

Figure 27: Constrained optimization algorithm

relations are encoded as constraints in the network as shown in Figure 25.

Figure 26 shows the KB structure and monotonicity constraints for pima-indian-diabetes. The task for this data set is to predict the risk of diabetes. This data set has 768 instances of which 268 are labeled positive. Domain knowledge suggests that an increase in each of the triceps' skin fold thickness (skin) is expected with an increase in the number of experienced pregnancies (preg), an increase in age (age), and perceived risk due to pedigree (pedi). The same monotonic relations are also suggested in body mass index (mass). An increase in preg, pedi, age, skin, or mass increases the risk of diabetes (class). Most diabetics have high levels of plasma glucose concentration (plas) and most suffer from high blood pressure (pres) while having low levels of insulin (insu).

We selected an implementation of the L-BFGS[6] algorithm from the conditional random fields package Mallet [33] to optimize our penalized objective function. To ensure convergence in the feasible region, the L-BFGS maximization was wrapped in the outer-level algorithm given in Figure 27. For running experiments, we integrated the learning algorithm with Weka, which allowed us to easily script learning runs and to run comparisons against other learning algorithms. The algorithms we analyzed were:

Zero-Regression (ZR) Always picks the mode of the observed distribution of the class variable, without regard to the features.

Naïve Bayes (NB) Also known as the simple Bayesian classifier (SBC). Treats the class variable as the parent in a Bayesian network, with all features as children.

[6]Limited-memory BFGS, a variation of the the *Broyden-Fletcher-Goldfarb-Shanno* algorithm (see, e.g., [38, pg.324]).

Knowledge-based Bayes (KB) Fit the parameters of a Bayesian network whose structure incorporates domain knowledge. Parameters are fit by maximum likelihood with a Laplace correction.

Constrained Knowledge-based Bayes (CKB) Same as KB, except that the parameters are fit to maximize the posterior probability subject to the inequality constraints induced by the qualitative monotonicity statements. CKB was run with three different margins, $\varepsilon \in \{0.0, 0.1, 0.2\}$. These runs are designated CKB0, CKB0.1, and CKB0.2.

To compare the algorithms on each data set, we first randomly split the data set into a test set (1/3 of the data) and a training set pool (2/3 of the data), stratified by class. Then we performed 50 replications for each training set size m, for various m from 1 to 50. In each replication, we randomly drew m elements without replacement from the training set pool, and trained our algorithms on the set. The resulting fitted networks were then evaluated on the test set. Plots for two of the data sets auto and pima are shown in Figure 28. In general, the experimental results indicate significant improvement in performance for networks that exploited monotonicity knowledge over those that did not.

Our hypothesis was that ZR would perform the worst, that Naïve Bayes would be the second worst, that the Knowledge-based networks would come next, and that the networks that combined knowledge-based structure with monotonicity constraints (CKB) would give the best results. The results indicate that actually, ZR performs surprisingly well: comparable to or better than NB at small sample sizes on all data sets. Otherwise, we do see the expected ranking, though at small sample sizes, ZR and NB frequently tie, as do KB and CKB.

Since our largest training set was of size 50, which we still consider relatively small for models of the complexity used here, we actually expected to see this ranking extend through more of the tested sample sizes. One somewhat surprising result was how well NB performed on one data set car (not shown here but see [1]) at higher sample sizes with discretizations finer than 2 bins. We were also particularly surprised by the results on the haberman data set, where NB and ZR did very well all the way through $m = 50$ (see [1] for additional details). A simple data analysis on the haberman data (2-bin) using correlation and mutual information revealed that the data set exhibits independence between the class variable and any one of the three parent attributes. Moreover, the conditional probability tables reveal that the parameters do not exhibit monotonicity; for example, the chance of surviving given that the patient is young is high but surprisingly the data also says that the chances of surviving given that the patient is old is also high. In this case, our assumptions about the structure and monotonicities were clearly incorrect.

A second hypothesis was that the monotonicity constraints might be incorrect and lead to poor performance at large sample sizes, particularly with $\varepsilon = 0.2$. The plots do show flatter learning curves for CKB with $\varepsilon > 0$, compared to CKB with no margin. However, without margins, CKB is comparable to or better than KB at nearly all tested sample sizes.

A third hypothesis, as mentioned, was that monotonicity constraints would help more at finer discretizations. The plots show some support for this; on auto (Figure 28), there is little difference between CKB0 and KB at the 2-bin discretization, and at higher discretizations and with more training data, CKB0 dominates KB. CKB0.1 on auto shows very good performance at the 5-bin discretization level. In addition, looking only at lower sample sizes, we observe this effect on the pima (Figure 28) data set. The results from the remaining data sets do little to support or disprove this hypothesis.

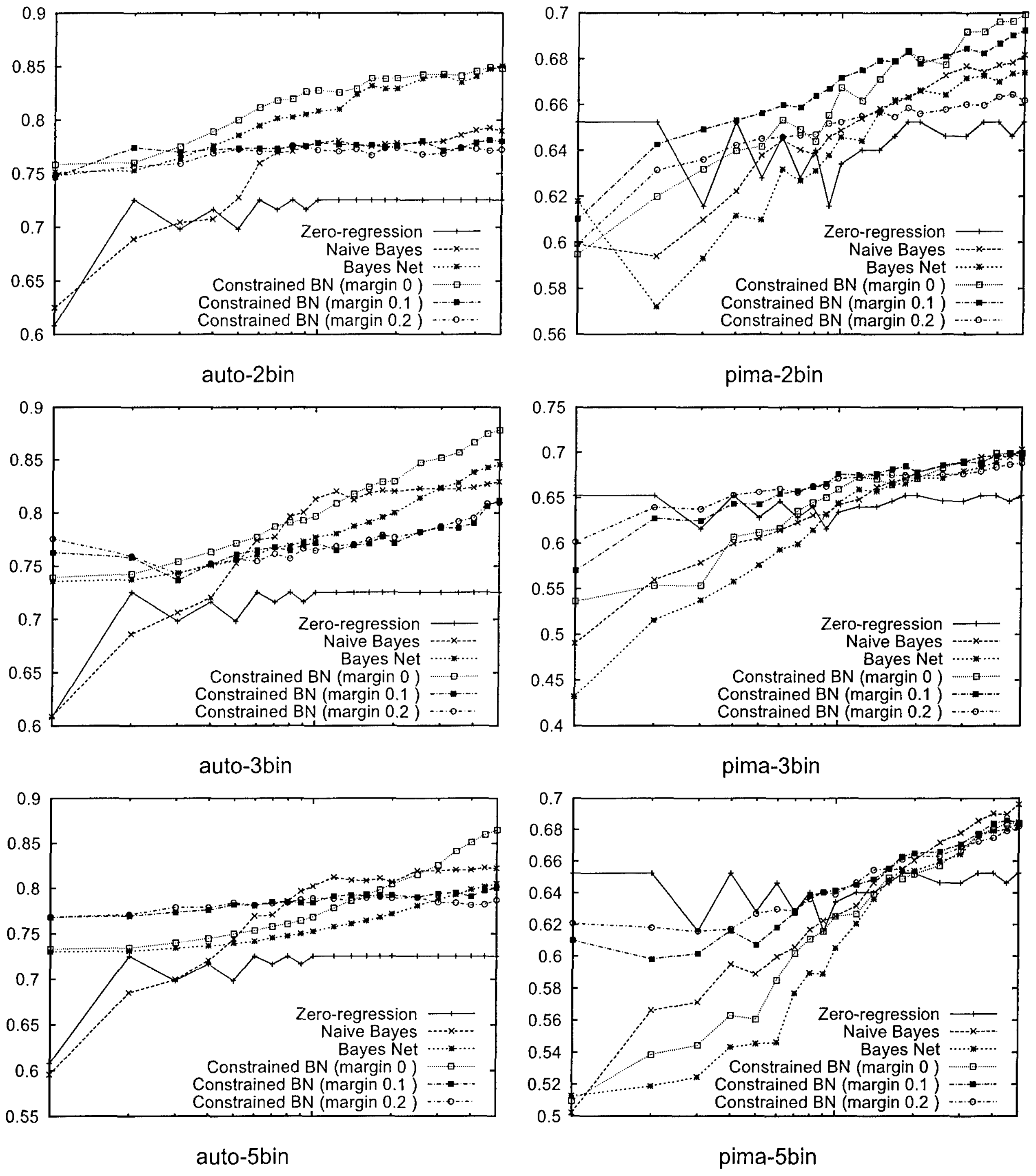


Figure 28: Learning curves for auto and pima domains at 3 discretizations, plotting average accuracy (across 50 runs) against training set size (log scale, 1 through 50).

In summary, we have shown a method for using qualitative domain knowledge in the estimation of parameters for a probabilistic model. The qualitative statements (monotonicity) are natural and easily specified by domain experts, and they have a probabilistic semantics consistent with a domain expert's intuition. We have shown that these semantics are tractable and can effectively constrain probability distributions, and that the constrained models in many cases generalize from the training set to the test set better than do unconstrained models. As mentioned previously in Section 3.1, we also want to additionally address the issue of parameter estimation as part of the multiple-parent problem in the case where the number of parents in Bayesian networks are bigger than the networks discussed in this section. Section 3.4.3 describes our work on constrained logistic regression that addresses this problem.

Data-Free Model Evaluation: Preliminary Results from a Prototype Evaluation

An important principle of machine learning is that the complexity of the learned model must be adapted to the amount of data that is available. This is normally achieved by defining a nested family of models of progressively greater complexity and using cross-validation methods to identify which member of the family gives the most accurate predictions. However, when data is scarce, cross-validation is not reliable. Hence, we sought to find a data-free method for evaluating whether the complexity of a model was well-matched to the amount of available data.

We designed a data-free model evaluator that would exploit the expert's rich domain knowledge captured by the Bayes net's full structure. The general idea is to determine, *without* using the training data the best Bayes net structure (perhaps a smaller variant of the initial structure) for a given small number of training samples. The hope is that by starting off from a structure that is the best for a given small training set size, learning with very few examples would yield better performance than using any other initial structure.

We designed prototype to test this idea. The prototype was given two network structures. One contains all of the dependencies that the domain expert provides (the "full structure"). The second structure is the structure that we wish to evaluate (the "eval structure"). The prototype produces a learning curve for the eval structure without consulting the training data. It does this by repeatedly 1) sampling network parameters for the full structure, 2) generating synthetic data based on the sampled parameters, and 3) creating a learning curve for the eval structure when learning from the synthetic data. Steps 1-3 are repeated many times and averaged to produce a final averaged learning curve.

We implemented a .NET prototype for a data-free model evaluator that uses a uniform distribution over CPT parameters. We ran initial experiments on one of the datasets (car) from our UAI 2005 paper [1] on monotonicity constraints. In the experiments, the full structure was the one used in the UAI 2005 paper and the eval structure was a simple naive Bayes network. The hope was that the learning curves would be qualitatively similar to the curves in the UAI 2005 paper. The results on this dataset were somewhat *inconclusive*. Here is a brief summary of the results. The procedure samples many network parameter settings and creates a learning curve for each of them and finally averages these learning curves.

1. The final averaged learning curves exhibit a "crossing point" at 3 training examples. That is, the

naive Bayes curve is superior until some number of training examples and then becomes inferior to the full model structure. In contrast, the experiments reported in UAI 2005 paper show that the crossing point is at about 10 training examples for the naive Bayes and full model.

2. Looking at the learning curves for individual parameter samplings, we see that while many of them do show crossing points within 80 training examples (the maximum in our experiments), many others do not have a crossing point. That is, often either the naive Bayes or full model dominate the other for all training set sizes up to 80 training examples.

3. Looking at the learning curves for the individual parameter samplings, we see that there is a wide range of crossing points. This shows that the particular number of examples where one model is better than the other depends strongly on the particular network parameters, and not just the structure.

4. An informal assessment indicates that even for parameter samplings with about the same Bayes rate, the crossing points can vary substantially. Thus, information about the Bayes rate and structure do not seem to be enough to zero in on accurate crossing points.

A drawback of our experiment is that it did not incorporate the monotonicity constraints that we included in the UAI paper. Those constraints should have caused the crossing point to occur even earlier than it would have occurred without the constraints. So the observation that the average crossing point was already very early suggests that our data-free evaluation strategy is not working correctly in its current form.

On a side note, we invented some computational mechanisms for speeding up the sampling process by using Bayes net inference. In particular, Bayes net inference can be used to reduce the sampling effort for computing the error rate of a learned model and the Bayes rate of the target model. These ideas were not incorporated into the evaluated prototype, but they could help speed up any future implementation of this approach.

3.4.3 Constrained Logistic Regression

We investigated possible ways to address the problem of fitting a model in which at least one node in the network has a large number of parents. This is a specific concern in our domain since our training sets are very small and the number of parameters could explode exponentially. For example, for discretized bin=5, a node with five parents could easily have a CPT with $5^6 = 15,625$ parameters to fit. The problem boils down to finding a way to keep the number of parameters from exploding. We have specifically looked at two techniques: *noisy-max* and *logistic regression*. Noisy-max (Henrion 1989) is simply the generalization of noisy-or. Noisy-or allows for the uncertainty that a parent affects the child, with the assumption that the inhibition of each parent is independent of that of any other parents. CPTs can then be constructed using only parameters linear in the number of parents k, *i.e.,* it only requires $O(k)$ instead of $O(2^k)$. Logistic regression, on the other hand, describes the log ratio of class-conditional densities as a linear function of the parents. For example,

$$\log \frac{p(C_1|X_1 = x_1, X_2 = x_2)}{1 - p(C_1|X_1 = x_1, X_2 = x_2)} \quad = \quad \beta_0 + \beta_1 X_1 + \beta_2 X_2 \qquad (16)$$

Here, the monotonicity constraints can be implemented as constraints on the signs of the parameters β_i. The qualitative influence $Q+$ translates to the constraint $\beta > 0$ and $Q-$ translates to the constraint $\beta < 0$. The advantage of using logistic regression is that it keeps the number of parameters from exploding and it allows incorporation of synergistic and anti-synergistic influences. We can take into consideration the effects of a subset of variables, assuming the domain expert can specify them. For example,

$$\log \frac{p(C_1|X_1 = x_1, X_2 = x_2)}{1 - p(C_1|X_1 = x_1, X_2 = x_2)} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 \tag{17}$$

Synergy can simply be translated as a constraint on β_3, *i.e.,* $\beta_3 > 0$. If X_1 and X_2 are anti-synergistic, then the constraint could be set as $\beta_3 < 0$, where $|\beta_3| \geq \min(\beta_1, \beta_2)$. When $\beta_3 = 0$, then we obtain pure additive effect as in (16). We have chosen this latter approach to address large numbers of parents and have called it *constrained logistic regression.*

As discussed in Section 3.4.1 (also see [1]), it has been shown that significant improvements in classification accuracy with very small amounts of training data (less than 10 examples) can be made by exploiting qualitative monotonicities. However, when the number of parents N increases (*e.g.,* $N > 7$) the approach reported in [1] suffers from two distinct disadvantages. The number of parameters that need to be estimated and the number of constraints on the parameters required to implement monotonicity both increase exponentially with N. The former could lead to underfitting hence producing models that give oversimplified hypotheses, while the latter could indirectly impose practical limits on the size of the problem that can be solved by a computer due to huge memory requirements. In this section, we address these limitations by formulating the problem of exploiting qualitative monotonicities in Bayesian network parameter learning as a constrained logistic regression problem. This novel formulation affords us two major advantages. Parameter estimation can now be viewed as a regression problem over a set of parameters whose size only grows linearly with the number of parents. Moreover, the number of constraints required to exploit qualitative monotonicity can be shown to only grow linearly with the number of parents as well.

Clearly, not all constrained logistic regression models address the problems outlined above. To illustrate our point, we show one model that has practical significance. Restificar and Dietterich [40] has a detailed treatment and analysis of constrained logistic regression models. In this report, however, we show only a variant of the logistic regression model in which the number of parameters needed to learn the conditional probability distributions in a Bayesian network as well as the number of constraints required to exploit qualitative monotonicities only grow linearly with the number of parents. Preliminary experimental results using constrained logistic regression indicate improved accuracy and performance over the technique discussed in Section 3.4.1, *i.e.,* on Bayesian nets with nodes having $8 - 11$ parents. Let us now define the constrained logistic regression model.

Defi nition 1 ($\mathcal{M}_{CLR}$) *Let Y be a child variable with k_Y levels, $k_Y \geq 2$, in a Bayesian network BN with parents $X_1, \ldots, X_n$ where each X_i has k_{X_i} levels, $k_{X_i} \geq 2$ for $i = 1, \ldots, n$. Given some configuration*

$c = \langle x_1, \ldots, x_n \rangle$, define

$$\log \frac{P(Y > j|c)}{P(Y \leq j|c)} = \beta_{0j} + \beta_{1j}^{X_1} I[X_1 = 1] + \cdots + \beta_{k_{X_1}j}^{X_1} I[X_1 = k_{X_1}] + \cdots + \beta_{1j}^{X_n} I[X_n = 1] + \cdots + \beta_{k_{X_n}j}^{X_n} I[X_n = k_{X_n}] \text{ for } j = 0, \ldots, k_Y - 2 \tag{18}$$

Also, let the following set of constraints hold

1. *For each parent X_i, $i = 1, \ldots, n$ and $j = 0, \ldots, k_Y - 1$*

$$\beta_{k_{X_i}j}^{X_i} \geq \beta_{(k_{X_i}-1)j}^{X_i} \geq \cdots \geq \beta_{2j}^{X_i} \geq \beta_{1j}^{X_i} \geq 0 \tag{19}$$

2. *For each pair $(j-1), j$, where $j = 1, \ldots, k_Y - 2$*

$$\begin{array}{rcl} \beta_{0(j-1)} & \geq & \beta_{0j} \\ \beta_{1(j-1)}^{X_1} & \geq & \beta_{1j}^{X_1} \\ & \vdots & \\ \beta_{k_{X_1}(j-1)}^{X_1} & \geq & \beta_{k_{X_1}j}^{X_1} \\ & \vdots & \\ \beta_{1(j-1)}^{X_n} & \geq & \beta_{1j}^{X_n} \\ & \vdots & \\ \beta_{k_{X_n}(j-1)}^{X_n} & \geq & \beta_{k_{X_n}j}^{X_n} \end{array} \tag{20}$$

The conditional distribution in the model is estimated by using the parent levels of the random variable. Here, we also estimate the conditional distribution by estimating $k_Y - 1$ logistic functions and then subtract the values of the estimated adjacent logistic functions to compute the conditional distribution of Y given a parent configuration. Definition 1 expresses the log odds ratio between the cumulative conditional probabilities $P(Y > j|c)$ and $P(Y \leq j|c)$ as a linear function of the levels of the parent variables $X_1, \ldots, X_n$. Such formulation allows one to model the contribution of each level of X_i to the log odds ratio via the β parameters. M_{CLR} also allows the expression of synergistic and anti-synergistic influences where terms that interact are simply added to the linear function. In addition, the number of constraints N_C for model M_{CLR} is linear in the number of parent levels since $N_C = n + \sum_{i=1}^{n}(k_{X_i} - 1) + (k_Y - 1)\sum_{i=1}^{n}(k_{X_i} - 1) = n + k_Y \sum_{i=1}^{n}(k_{X_i} - 1)$. The number of β parameters, N_β, is also linear in the number of parent levels, $N_\beta = (k_Y - 1)[1 + \sum_{i=1}^{n}(k_{X_i} - 1)]$. The following theorem states that the constraints we impose on the β parameters of M_{CLR} are sufficient for first-order stochastic dominance. The proof of the theorem is given in [40].

Theorem 1 *Let Y be a child variable with k_Y levels, $k_Y > 2$, in a Bayesian network BN with parents $X_1, \ldots, X_n$ where each X_i has k_{X_i} levels, $i = 1, \ldots, n$. If M_{CLR} is the constrained logistic regression model, $Q+$ the qualitative influence of each X_i on Y, then Y is* **FSD** *monotonic in X_i.*

To evaluate our model, we have performed several experiments. We describe the experimental setup and preliminary results below.

Preliminary Experimental Results

We conducted a series of experiments using the same setup and included the same datasets described in Section 3.4.2. We used a total of ten datasets. The additional five datasets were the previously discarded "large" datasets from the experiments in Section 3.4.2. The networks for these datasets have nodes with incoming arcs that number between $8-11$, *i.e.,* networks which were not well addressed by the previous method. In addition to the algorithms used in the previous experiments, the following constrained logistic regression algorithms were added:

- $ZLR(\beta_1, \beta_2)$: constrained logistic regression *without* Laplace correction using β_1 and β_2 margins
- *ZLN*: *ZLR* with no margins
- $CLR(\beta_1, \beta_2)$: constrained logistic regression *with* Laplace correction using β_1 and β_2 margins
- *CLN*: *CLR* with no margins

β_1 and β_2 enforce inequality margins for constrained logistic regression in the same way that the ε in Equation (15) does for the previous technique. Assuming a parent variable X_i with levels $0,1,2$, the β_1 margin is a constraint placed upon the β parameter associated with the indicator function $I[X_i = 1]$ in Equation (18). When X_i changes value from 0 to 1, we want the sum of the βs on the right-hand side of Equation (18) to increase by at least β_1. In addition, when X_i changes value from 1 to 2, we want the the sum of the βs on the right-hand side of Equation (18) to increase by β_2. The plots for the datasets auto and pima are shown in Figure 29. Plots for three of the datasets (postop, housing, and nursery) with networks that have nodes with number of parents ranging from $8-11$ are shown in Figure 30.

Preliminary results indicate that constrained logistic regression can significantly improve average classification accuracy over the previously reported technique by Altendorf et al. [1]. These are supported by plots shown in Figure 29 and Figure 30. Except for the nursery dataset in which Naïve Bayes is dominant and at lower discretizations ($bin = 2$) in Figure 29 in which Constrained BN (*CBN*) is better than all the other algorithms, constrained logistic regression appears to be the dominant algorithm. However, there seems to be no clear specific variant (*i.e.,* with different margins) of constrained logistic regression that consistently performs well across the ten datasets. There is, however, a marked difference between using variants of *CLR* and *ZLR*. *ZLR* (including *ZLN*) appears to perform better on "large" datasets than on datasets whose networks have parents that number less than 8. The opposite can be observed about *CLR* and *CLN*. We are currently running additional experiments to test other techniques that would allow us to automatically determine the best choice for the specific variant of the contrained logistic regression algorithm under conditions of sparse training data.

3.5 Publications

The following papers and technical reports have been produced either in part or with full support of the KILEARN research grant.

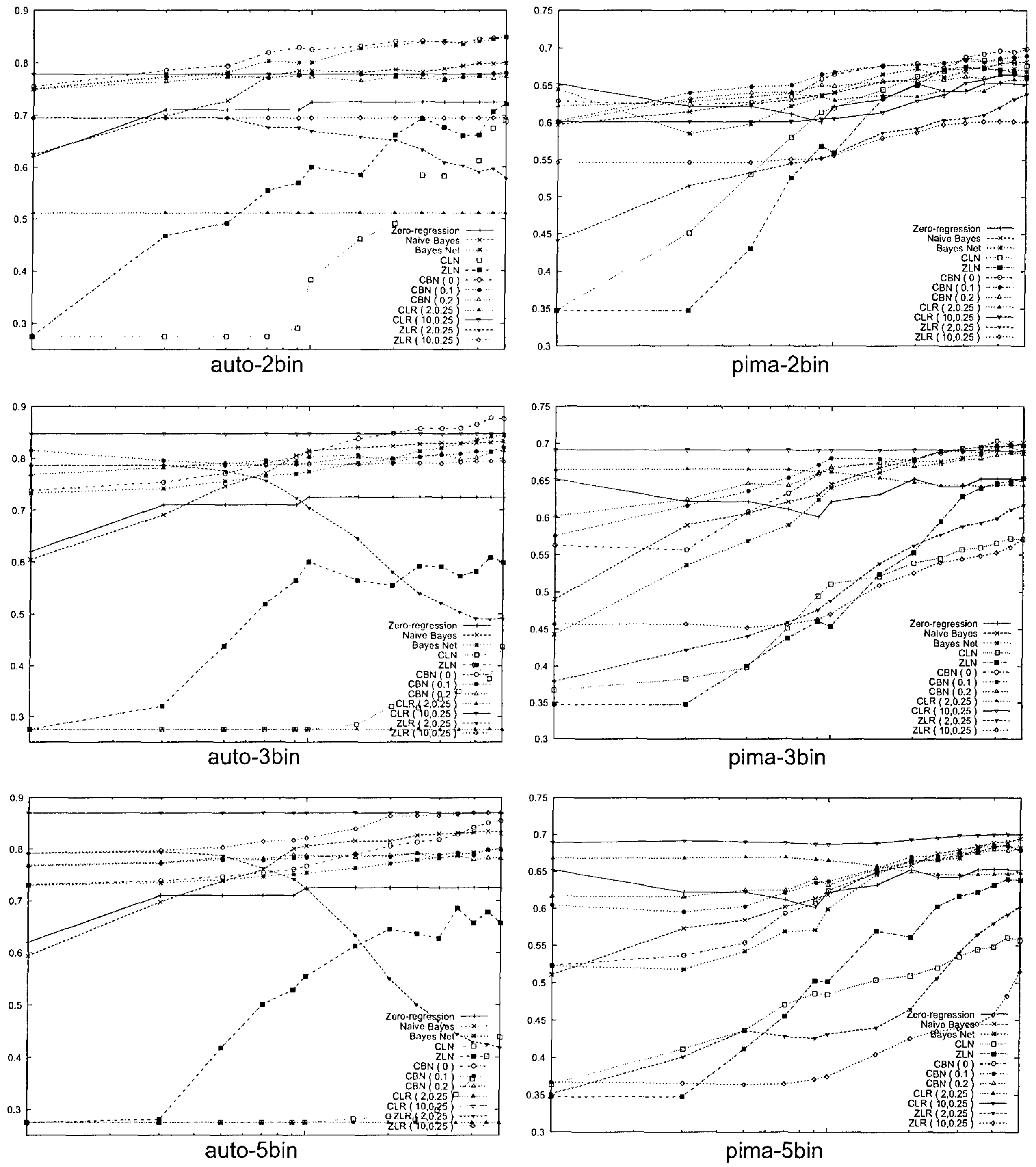


Figure 29: Learning curves using constrained logistic regression for auto and pima domains at 3 discretizations, plotting average accuracy (across 50 runs) against training set size (log scale, 1 through 50).

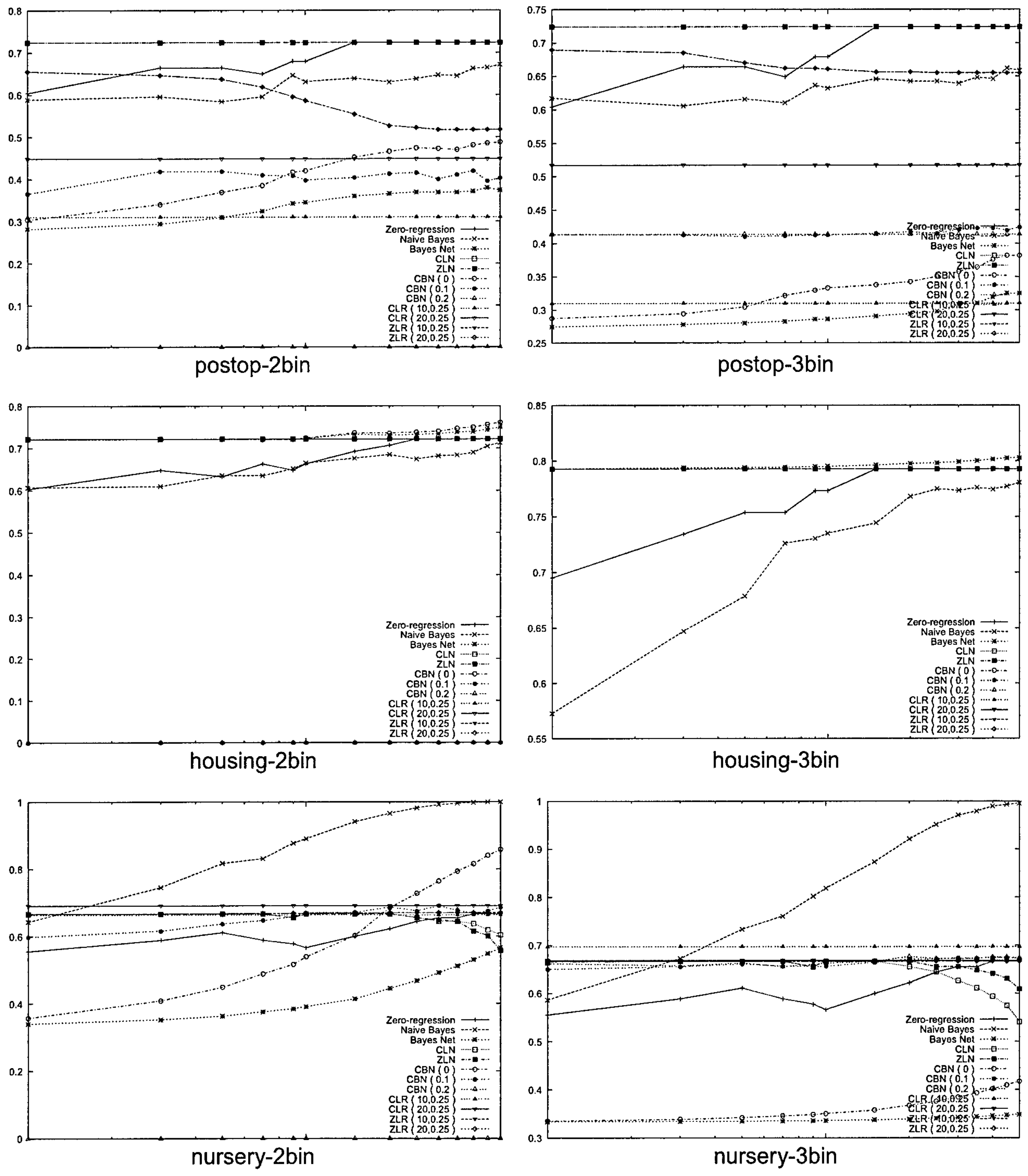


Figure 30: Learning curves using constrained logistic regression for postop, housing and nursery domains at 2 discretizations, plotting average accuracy (across 50 runs) against training set size (log scale, 1 through 50).

3.5.1 Conferences and Workshops

1. A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, M. Li, and J. L. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of the 2005 International Conference on Intelligent User Interfaces (IUI 2005)*, San Diego, California, January 2005.

2. Eric E. Altendorf, Angelo C. Restificar, and Thomas G. Dietterich. Learning from sparse data by exploiting monotonicity constraints. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005)*, Edinburgh, Scotland, July 2005.

3. Sriraam Natarajan, Prasad Tadepalli, Eric Altendorf, Thomas G. Dietterich, Alan Fern, and Angelo Restificar. Learning first-order probabilistic models with combining rules. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*, Bonn, Germany, August 2005.

4. J. Shen, L. Li, T. Dietterich, and J. Herlocker. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *Proceedings of the 2006 International Conference on Intelligent User Interfaces (IUI 2006)*, Sydney, Australia, January 2006.

5. X. Bao, J. Herlocker, and T. Dietterich. Fewer Clicks and Less Frustration: Reducing the Cost of Reaching the Right Folder. In *Proceedings of the 2006 International Conference on Intelligent User Interfaces (IUI 2006)*, Sydney, Australia, January 2006.

3.5.2 Technical Reports

1. Sriraam Natarajan and Eric Altendorf. First-Order Conditional Influence Language. Technical Report CS05-30-01, Oregon State University, School of Electrical Engineering and Computer Science, 2005.

2. Angelo C. Restificar and Thomas G. Dietterich. Exploiting Monotonicity via Logistic Regression in Bayesian Network Learning. Technical Report CS06-30-01, Oregon State University, School of Electrical Engineering and Computer Science, 2006.

3.6 Online Materials

More details about the conference papers, technical reports, and software (executable binaries) that were produced as a result of this project (either in part or with full support from this grant) can be found at the following website:

`http://web.engr.oregonstate.edu/~tgd/ki-learn`

4 Summary and Conclusion

Our goal in this research effort was to develop a new methodology, called KI-LEARN (Knowledge Intensive **LEARN**ing), that combines domain knowledge and sparse training data to construct high-performance systems. This technical report gave an overview of the major results we have obtained through the KI-LEARN research grant:

- We designed and implemented a language called FOCI (First-Order Conditional Influence) for expressing objects, relations, and attributes relevant to learning. Our language extends probabilistic relational models (PRMs), which are the probabilistic representations most similar to the first-order representation languages employed in KRR systems. A distinct feature of our language is its support for explicit expression of qualitative constraints such as monotonicity, saturation, and synergies. The primary goal of the language design was not to come up with the most expressive language but to design the most useful and tractable knowledge representation tool suitable for efficient learning.

- We developed learning algorithms for learning the parameters of combining rules in FOCI. These combining rules are significantly more expressive than aggregators in PRMs. They can be combined with aggregators to capture probabilistic relationships involving variable numbers of parent nodes in a bayesian network.

- We analyzed several domains and represented them using FOCI and PRMs. This analysis showed that FOCI has sufficient expressive power to capture all of these domains. In addition, it showed that qualitative monotonicity constraints arise in many domains.

- We developed algorithms for learning with qualitative monotonicity constraints. One algorithm learned Bayesian network conditional probability tables with the standard parameterization. The other algorithm expressed the conditional probability distributions as logistic regression functions. We experimented with two methods for enforcing the constraints: (a) penalty functions and (b) using modern constrained optimization packages. Our results show that incorporating monotonicity constraints often improves the speed and accuracy of learning.

In this report, we also described the models we constructed for our testbed domains and gave a description of the infrastructure we built for the task-based user interface domain as well as further improvements we made to the software prototype. Finally, we provided a list of research publications that were produced either in part or solely through the KI-LEARN research grant.

References

[1] Eric E. Altendorf, Angelo C. Restificar, and Thomas G. Dietterich. Learning from sparse data by exploiting monotonicity constraints. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005)*, Edinburgh, Scotland, July 2005.

[2] Norman P. Archer and Shouhong Wang. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, 24(1):60–75, 1993.

[3] X. Bao, J. Herlocker, and T. Dietterich. Fewer clicks and less frustration: Reducing the cost of reaching the right folder. In *Proceedings of the 2006 International Conference on Intelligent User Interfaces*, Sydney, Australia, January 2006.

[4] Arie Ben-David. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19(1):29–43, 1995.

[5] Kristin P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

[6] John Binder, Daphne Koller, Stuart Russell, and Keiji Kanazawa. Adaptive Probabilistic networks with hidden variables. *Mach. Learn.*, 29(2-3):213–244, 1997.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occams razor. *Inform. Proc. Lett.*, 24:377–380, April 1987.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[9] D. G. Bobrow, editor. *Qualitative Reasoning about Physical Systems*. MIT Press, Cambridge, MA, 1985.

[10] M. Bohanec and V. Rajkovic. Knowledge acquisition and explanation for multi-attribute decision making. In *Proc. Intl. Workshop on Expert Systems and their Applications*, pages 59–78, 1988.

[11] Peter Clark and Stan Matwin. Using qualitative models to guide inductive learning. In *Proc. International Conference on Machine Learning*, pages 49–56, 1993.

[12] H. Daniels, A. Feelders, and M. Velikova. Integrating economic knowledge in data mining algorithms. In *Intl. Conf. Soc. Comp. Economics*, 2002.

[13] H. Daniels and B. Kamp. Application of MLP networks to bond rating and house pricing. *Neural Computing & Applications*, 8:226–234, 1999.

[14] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society*, B.39, 1977.

[15] F. Densel, F. Guinchiglia, and D. McGuiness, editors. *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning*, Toulouse, France, 2002. Morgan Kaufmann.

[16] Thomas G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18(4):97–136, 1997.

[17] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, M. Li, and J. L. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of the 2005 International Conference on Intelligent User Interfaces*, San Diego, California, January 2005.

[18] Ramez A. Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[19] A. J. Feelders. Prior knowledge in economic applications of data mining. In Djamel A. Zighed, Henryk Jan Komorowski, and Jan M. Zytkow, editors, *PKDD*, volume 1910 of *Lecture Notes in Computer Science*, pages 395–400. Springer, 2000.

[20] K. D. Forbus. Qualitative process theory. In D. G. Bobrow, editor, *Qualitative Reasoning about Physical Systems*, pages 85–168. MIT Press, Cambridge, MA, 1985.

[21] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Relational Data Mining*. Springer-Verlag, 2001.

[22] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. *Invited contribution to the book Relational Data Mining, S. Dzeroski and N. Lavrac, Eds.*, 2001.

[23] S. J. Haberman. Generalized residuals for log-linear models. In *Proc. 9th International Biometrics Conference*, pages 104–122, 1976.

[24] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[25] David Heckerman, Christopher Meek, and Daphne Koller. Probabilistic Models for Relational Data. Technical Report MSR-TR-94-08, Microsoft Research, 2004.

[26] Herbert Kay and Lyle Ungar. Deriving monotonic function envelopes from observations. In *Proc. Qualitative Reasoning about Physical Systems*, 1993.

[27] K. Kersting and L. De Raedt. Basic principles of learning bayesian logic programs, 2002.

[28] Kristian Kersting and Luc De Raedt. Bayesian logic programs. In *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, 2000.

[29] Jörg-Uwe Kietz and Stefan Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In Stephen Muggleton, editor, *Proc. of Int. Workshop on Inductive Logic Programming*, Viana de Castelo, Portugal, 1991.

[30] Benjamin Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, 1994.

[31] N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, 1994.

[32] E. L. Lehmann. Ordered families of distributions. *Annals of Mathematical Statistics*, 26:399–419, 1955.

[33] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[34] Sriraam Natarajan and Eric Altendorf. First-Order Conditional Influence Language. Technical Report CS05-30-01, Oregon State University, School of Electrical Engineering and Computer Science, 2005.

[35] Sriraam Natarajan, Prasad Tadepalli, Eric Altendorf, Thomas G. Dietterich, Alan Fern, and Angelo Restificar. Learning first-order probabilistic models with combining rules. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*, Bonn, Germany, August 2005.

[36] Liem Ngo and Peter Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science (Selected Papers from the International Workshop on Uncertainty in Databases and Deductive Systems)*, 171:147–177, 1997.

[37] Rob Potharst and A. J. Feelders. Classification trees for problems with monotonicity constraints. *SIGKDD Explorations*, 4(1):1–10, 2002.

[38] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical recipes in C: the art of scientific computing.* Cambridge University Press, 1988.

[39] R. Quinlan. Combining instance-based and model-based learning. In *Proc. International Conference on Machine Learning*, pages 236–243, 1993.

[40] Angelo C. Restificar and Thomas G. Dietterich. Exploiting Monotonicity via Logistic Regression in Bayesian Network Learning. Technical Report CS06-30-01, Oregon State University, School of Electrical Engineering and Computer Science, 2006.

[41] J. Shen, L. Li, T. Dietterich, and J. Herlocker. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *Proceedings of the 2006 International Conference on Intelligent User Interfaces*, Sydney, Australia, January 2006.

[42] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proc. Computer Applications and Medical Care*, pages 261–265, 1988.

[43] Ryszard Szekli. *Stochastic Ordering and Dependence in Applied Probability.* Springer-Verlag, 1995.

[44] Linda C. van der Gaag, Hans L. Bodlaender, and Ad Feelders. Monotonicity in Bayesian networks. In *Proc. UAI-04*, pages 569–576, Arlington, Virginia, 2004. AUAI Press.

[45] M. P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artif. Intell.*, 44(3):257–303, 1990.

[46] Ian H. Witten and Eibe Frank. *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, 2000.

[47] Jennifer Orme Zavaleta, Jane Jorgensen, Bruce DAmbrosio, Hans K. Luh, Fredrick W. Kutz, and Philippe A. Rossignol. Data-driven discovery and interactive development of community level model of disease transmission: West Nile Virus in Maryland. Technical report, Oregon State University, Department of Computer Science, 2003.

[48] B. Zupan, M. Bohanec, I. Bratko, and J. Demsar. Machine learning by function decomposition. In *Proc. International Conference on Machine Learning*, 1997.